UNIVERSIDAD POLITÉCNICA DE MADRID

E.U. I.T. TELECOMUNICACIÓN

MEMORIA PROYECTO FIN DE CARRERA

RECONOCIMIENTO FACIAL

Alumno: Juan Alfonso Urtiaga Abad

Tutor: José Luis Rodríguez Vázquez

Septiembre 2014

Agradecimientos

Ahora que por fin ha llegado la hora de dar el último paso, quiero agradecer a mi familia el apoyo y la comprensión que me han dado todo estos años, y también agradecerles que me animaran a estudiar una carrera para poder tener un futuro mejor.

En especial, quiero darles las gracias a mis padres, que siempre han estado ahí durante todos los años de estudio, animándome a seguir adelante.

A mis compañeros y compañeras de clase, decirles que ha sido un placer haberles conocido, y que no vamos a perder el contacto por muchos kilómetros que se hagan para buscar trabajo.

A mis profesores, agradecerles la ilusión que ponen a la hora de enseñar, ya que ser profesor es algo más que dar una clase magistral, es animar a sus alumnos a que quieran saber más y más.

A mis amigos y en general a la gente que me rodea, agradecerles que aunque pasen los años y pase mucho tiempo sin que hablemos, siempre están y estarán ahí, y yo también para ellos.

Y por último, a mi novia, que dentro de un año será mi mujer, agradecerle lo mucho que me ha aguantado todo este tiempo y lo mucho que me ha ayudado. Ya no hará falta que me digas más “cari, ponte con el proyecto”.

El presente proyecto trata sobre uno de los campos más problemáticos de la inteligencia artificial, el reconocimiento facial. Algo tan sencillo para las personas como es reconocer una cara conocida se traduce en complejos algoritmos y miles de datos procesados en cuestión de segundos.

El proyecto comienza con un estudio del estado del arte de las diversas técnicas de reconocimiento facial, desde las más utilizadas y probadas como el PCA y el LDA, hasta técnicas experimentales que utilizan imágenes térmicas en lugar de las clásicas con luz visible.

A continuación, se ha implementado una aplicación en lenguaje C++ que sea capaz de reconocer a personas almacenadas en su base de datos leyendo directamente imágenes desde una webcam. Para realizar la aplicación, se ha utilizado una de las librerías más extendidas en cuanto a procesado de imágenes y visión artificial, OpenCV. Como IDE se ha escogido Visual Studio 2010, que cuenta con una versión gratuita para estudiantes.

La técnica escogida para implementar la aplicación es la del PCA ya que es una técnica básica en el reconocimiento facial, y además sirve de base para soluciones mucho más complejas. Se han estudiado los fundamentos matemáticos de la técnica para entender cómo procesa la información y en qué se datos se basa para realizar el reconocimiento.

Por último, se ha implementado un algoritmo de testeo para poder conocer la fiabilidad de la aplicación con varias bases de datos de imágenes faciales. De esta forma, se puede comprobar los puntos fuertes y débiles del PCA.

This project deals with one of the most problematic areas of artificial intelligence, facial recognition. Something so simple for human as to recognize a familiar face becomes into complex algorithms and thousands of data processed in seconds.

The project begins with a study of the state of the art of various face recognition techniques, from the most used and tested as PCA and LDA, to experimental techniques that use thermal images instead of the classic visible light images.

Next, an application has been implemented in C + + language that is able to recognize people stored in a database reading images directly from a webcam. To make the application, it has used one of the most outstretched libraries in terms of image processing and computer vision, OpenCV. Visual Studio 2010 has been chosen as the IDE, which has a free student version.

The technique chosen to implement the software is the PCA because it is a basic technique in face recognition, and also provides a basis for more complex solutions. The mathematical foundations of the technique have been studied to understand how it processes the information and which data are used to do the recognition.

Finally, an algorithm for testing has been implemented to know the reliability of the application with multiple databases of facial images. In this way, the strengths and weaknesses of the PCA can be checked.

Tabla de contenido

1 LISTADO DE ACRÓNIMOS

BSD *Berkeley Software Distribution*

Distribución de Software Berkeley. Sistema operativo derivado de Unix y nacido gracias a los aportes de la Universidad de California en Berkeley.

C++

Lenguaje de programación diseñado para extender el lenguaje C con mecanismos que permitan la manipulación de objetos.

CSV *Comma Separated Values*

Valores separados por comas. Tipo de documento utilizado para representar datos en formato de tabla, en las que las columnas se separan por coma o por punto y coma.

IDE *Integrated Development Environment*.

Entorno de Desarrollo Integrado. Un entorno de desarrollo integrado es un programa informático compuesto por un conjunto de herramientas de programación. Puede dedicarse en exclusiva a un solo lenguaje de programación o bien puede utilizarse para varios.

IEEE. *Institute of Electrical and Electronics Engineers*.

Instituto de Ingenieros Eléctricos y Electrónicos. Es una asociación mundial de técnicos e ingenieros dedicada a la estandarización y el desarrollo en áreas técnicas.

LDA. *Linear Discriminant Analysis.*

Análisis Discriminante Lineal. Técnica estadística con el objetivo de identificar grupos de datos y discriminar entre los miembros de dichos grupos.

OpenCV. *Open Source Computer Vision Library.*

Librería para Visión por Computador de Código Abierto. Es una librería para visión artificial, de libre distribución y multiplataforma, desarrollada originalmente por Intel.

PCA. *Principal Components Analysis.*

Análisis de Componentes Principales. Técnica estadística utilizada para reducir la dimensionalidad de un conjunto de datos. En procesado de imagen es utilizada principalmente como herramienta de compresión y/o detección de patrones.

2 INTRODUCCION

2.1 Planteamiento

El presente proyecto tienes dos objetivos: realizar un estudio del arte de las diferentes tecnologías aplicadas al reconocimiento facial, y desarrollar una aplicación en lenguaje C++ que reconozca a un sujeto previamente almacenado en su base de datos, ayudándose para ello en librerías de código abierto para procesado de imágenes.

Para el estudio del arte han sido consultados varios libros de referencia sobre reconocimiento facial, donde se detallan análisis matemáticos, algoritmos, transformaciones, etc., así como diversos artículos del IEEE (*Institute of Electrical and Electronics Engineers* o Instituto de Ingenieros Eléctricos y Electrónicos) en el que se explican métodos más experimentales que nos pueden dar una idea de futuras alternativas más robustas y fiables.

En cuanto a la aplicación, la técnica de reconocimiento facial elegido ha sido el reconocimiento mediante PCA (*Principal Components Analysis* o Análisis de Componentes Principales), un método básico para entender algoritmos más avanzados, ya que la mayoría de ellos están basados de un modo u otro en este. Para su implementación se ha elegido la librería OpenCV al ser una de las librerías de libre distribución más extendidas y, por lo tanto, con más información disponible.

2.2 Concepto

El reconocimiento facial es uno de los grandes retos de la visión por ordenador. Algo tan sencillo para los seres humanos como identificar una cara conocida, en un sistema de visión artificial se convierte en un proceso matemático complejo y poco eficiente. Pequeños detalles para nosotros como el cambio de posición de un foco o un gesto momentáneo, son puntos críticos en cualquier sistema que pretenda conseguir una tasa de error baja.

Todo algoritmo de reconocimiento facial tienes dos fases bien diferenciadas, la fase de entrenamiento y la fase de reconocimiento o test.

Durante la fase de entrenamiento se prepara la base datos que será utilizada en el reconocimiento. Para ello, se introducen una o varias imágenes del rostro de los distintos sujetos en el algoritmo de entrenamiento. Este algoritmo es el encargado de extraer las características de cada persona y almacenarlas para su posterior comparación.

En la fase de test se toman imágenes de un sujeto desconocido, se extraen las características por el mismo proceso que en la fase de entrenamiento, y se comparan con las de la base de datos. El proceso consta de las siguientes partes:

- Toma de imagen: Dependiendo del sistema empleado, podrá ser una imagen fija, un fotograma de vídeo, una imagen tridimensional, etc.
- Detección de cara: La detección de una cara en una imagen no es más que un caso específico de detección de objetos. Hay que tener en cuenta si se van a reconocer una o varias caras a la vez.
- Procesado de la imagen: Una vez tengamos determinada la posición de la cara, la extraeremos de la imagen ya que el resto de la información es irrelevante. Procesar entera la imagen original solo serviría para aumentar la cantidad de cálculos y, por lo tanto, disminuir el rendimiento del sistema. También pueden ser necesarios otros procesos, como por ejemplo convertir la imagen a escala de grises, o utilizar un filtro paso bajo si la resolución de la imagen es muy alta.
- Extracción de las características: Una vez la imagen procesada, se calcularán los valores o coeficientes característicos de la imagen dependiendo de la técnica utilizada.
- Reconocimiento: Por último, se comparan los datos extraídos con los de la base de datos para encontrar al sujeto con los valores más próximos. Igual que antes, esta comparación dependerá de la técnica empleada.

La figura 1 es el diagrama correspondiente al diagrama de flujo previamente detallado.

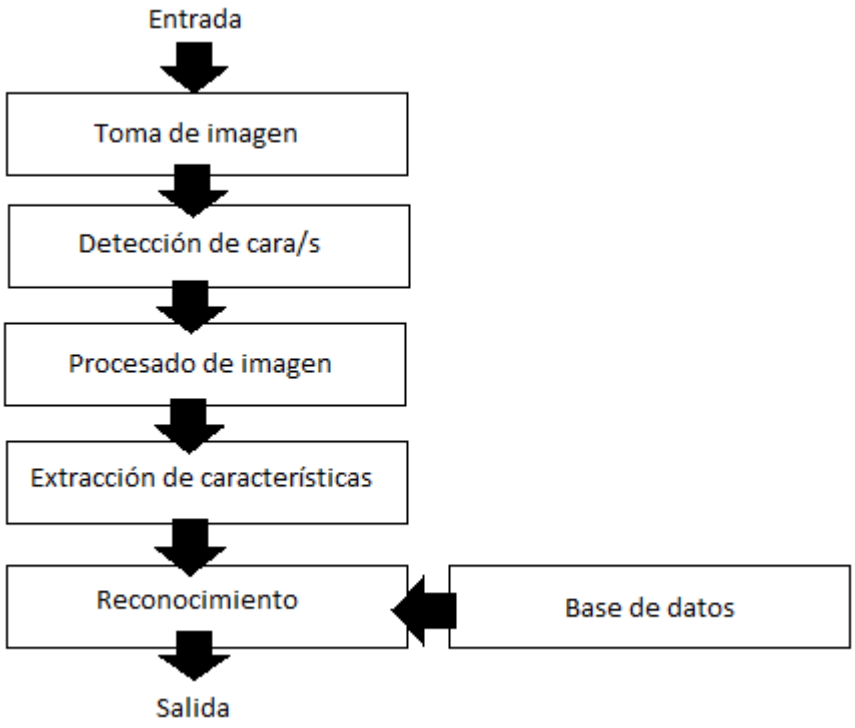


Figura 1 Proceso de reconocimiento facial

3 MARCO TECNOLÓGICO

Este capítulo se divide en tres partes. La primera hace referencia a la detección de caras en imágenes faciales, la segunda a las distintas técnicas y algoritmos de reconocimiento facial, y la tercera a las distintas librerías disponibles para procesado de imagen.

3.1 Detección facial

Lo primero que se ha de hacer en un sistema de reconocimiento facial es detectar una cara en una imagen. Esto no deja de ser un caso concreto de reconocimiento de patrones. Uno de los algoritmos más usados para detectar objetos en tiempo real es el detector rápido de objetos de Viola-Jones [1].

Desarrollado en 2001, actualmente es uno de los algoritmos más utilizados en detección de caras. El método de Viola-Jones es un método de aproximación basado en la apariencia. Está dividido en dos etapas: una primera etapa de aprendizaje del clasificador basada en un gran número de ejemplos positivos (los objetos de interés, como por ejemplo las caras) y de ejemplos negativos, y una fase de detección mediante la aplicación de este clasificador a las imágenes no conocidas.

En lugar de trabajar directamente con los valores de los píxeles, Viola y Jones proponen basar el algoritmo en características simples. Según los autores, la principal razón de esto es que las características pueden ser utilizadas para codificar conocimientos que sería mucho más difícil de hacerlo utilizando una cantidad finita de datos de entrenamiento. Además, al querer hacer la detección de objetos en tiempo real, hay una segunda ventaja en utilizar las características, y es que los sistemas basados en características son mucho más rápidos que los basados en píxeles.

3.1.1 Características Haar e imagen integral

Las características usadas en el algoritmo son las características tipo *Haar*. Estas características se calculan como la diferencia de la suma de los píxeles de dos o más zonas rectangulares adyacentes. En este algoritmo se utilizan tres tipos de características *Haar* que aparecen representadas en la figura 2:

- La característica-dos-rectángulos, que es la diferencia entre la suma de los píxeles de dos rectángulos. Estos rectángulos tienen la misma forma y son adyacentes vertical u horizontalmente.
- La característica-tres-rectángulos, que calcula la suma de los píxeles dentro de dos rectángulos exteriores, a su vez restados de la suma de un tercer rectángulo interior.
- La característica-cuatro-rectángulos, que calcula la diferencia de rectángulos pareados en diagonal.

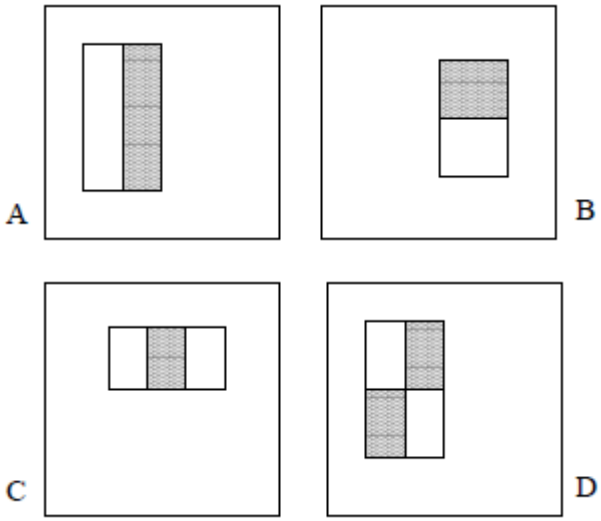


Figura 2 Características tipo Haar. A la suma de los píxeles marcados en gris se le resta la suma de los rectángulos marcados en blanco.

Para calcular las características rectangulares rápidamente, los autores proponen una imagen intermedia llamada imagen integral, que es una representación en forma de imagen del mismo tamaño que la imagen original, donde cada uno de los puntos se calcula como la suma de los píxeles situados por encima de él y a su izquierda. En la ecuación (1), ii sería la imagen integral e i la imagen original.

$$ii(x,y) = \sum_{x\leq x',y\leq y'} i(x',y') \quad \textbf{(1)}$$

Para implementar (1), se utilizan el siguiente par de ecuaciones:

$$s(x,y) = s(x,y-1) + i(x,y) \quad \textbf{(2)}$$

$$ii(x,y) = ii(x-1,y) + s(x,y) \quad \textbf{(3)}$$

donde s(x,y) es la suma acumulativa de las filas, siendo s(x,-1=0) e ii(-1,y)=0. Con este par de ecuaciones se puede calcular la imagen integral en una sola pasada de la imagen original.

Con la imagen integral ya calculada, cualquier suma rectangular puede ser calculada utilizando cuatro puntos de la matriz. En el siguiente ejemplo, el valor del punto 1 es el de la suma de todos los píxeles de la zona A, el de 2 es A + B, el de 3 es A + C, y el de 4 es A + B + C + D, por lo que si queremos conocer la suma de los píxeles dentro del rectángulo D, se puede calcular cómo 4+1-(2+3). La figura 3 es una representación de este ejemplo.

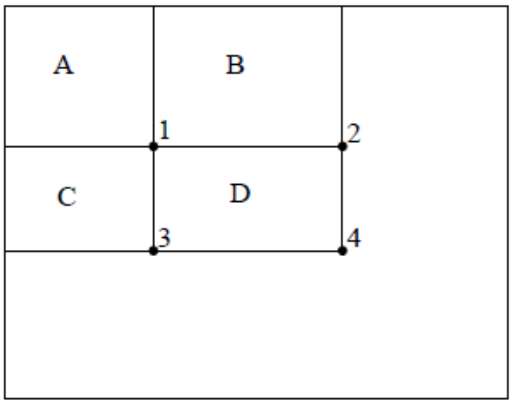


Figura 3 Ejemplo de uso de la imagen integral

3.1.2 Entrenamiento del clasificador

En el entrenamiento del clasificador de Viola-Jones, se utiliza una variante del algoritmo de AdaBoost para elegir un pequeño grupo de características y entrenar el clasificador. La idea del algoritmo es generar un clasificador preciso a partir de clasificadores más débiles. El algoritmo añade clasificadores simples uno tras otro. A continuación realiza una selección de características de entrenamiento de cada uno de ellos y los combina para conseguir un clasificador mucho más preciso.

En la figura 4, podemos ver la primera y segunda característica elegidas por AdaBoost. Las características aparecen en la primera fila, mientras que en la segunda aparecen superpuestas a una imagen facial típica. La primera característica mide la diferencia entre la región de los ojos y la de la parte superior de las mejillas. La segunda característica compara la intensidad en la región de los ojos con la del puente de la nariz.

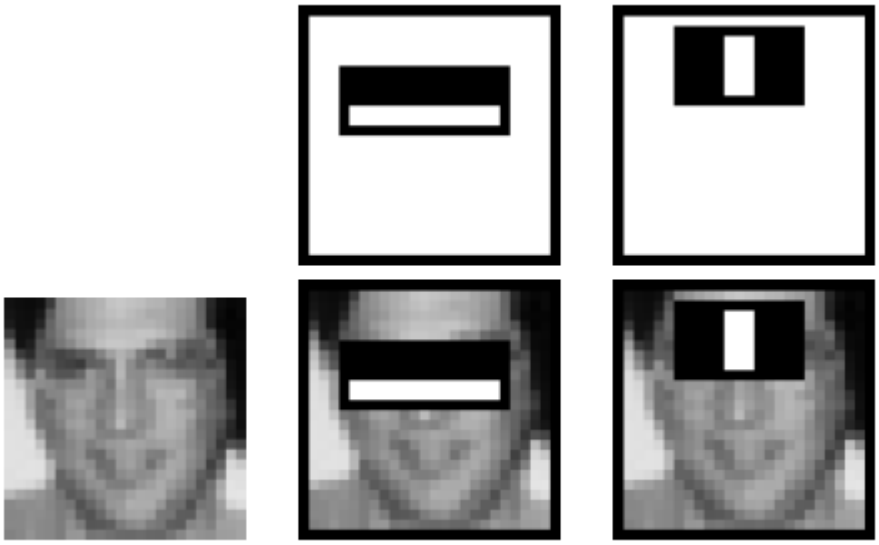


Figura 4 Ejemplo características elegidas por AdaBoost.

3.1.3 Cascada de clasificadores

Para finalizar, Viola y Jones proponen un esquema basado en una cascada de clasificadores para realizar la detección en tiempo real.

Cada etapa de la cascada corresponde a un clasificador, y está entrenada con todos los ejemplos que la etapa anterior no ha clasificado correctamente más algunos nuevos. En la fase de entrenamiento, cada etapa se entrena con un conjunto óptimo de características capaces de detectar cada vez ejemplos más complicados respecto a los de la etapa anterior; es decir, las primeras etapas se encargan de descartar sub-imágenes que son muy diferentes de una cara, mientras que las últimas etapas pueden rechazar ejemplos mucho más complejos como pueden ser globos, pelotas, dibujos, etc...

De esta forma, se busca optimizar el algoritmo para conseguir un índice alto de positivos y bajo de falsos positivos, sin tener que incrementar el coste computacional, algo crítico en un sistema en tiempo real.

3.2 Reconocimiento facial. Técnicas basadas en imágenes fijas

3.2.1 PCA

[2] El Análisis de Componentes Principales es una técnica estadística útil tanto para la compresión de imágenes como en el campo de reconocimiento facial. El uso del PCA en el procesado de imágenes faciales es muy habitual, por lo que es una técnica de la que se puede encontrar mucha información. Es la técnica elegida para implementar en el presente proyecto al ser básica para entender el reconocimiento facial. Además, sigue siendo una parte de importante de técnicas más avanzadas.

Esta técnica se basa en dos fases, una de entrenamiento y otra de clasificación. En la fase de entrenamiento, y por medio del PCA, se forma el espacio de facciones (*eigenspace*) a partir de las imágenes faciales de entrenamiento. El espacio de facciones es la matriz formada por los *eigenvectors* (vectores propios). Estos vectores contienen la información de la variación de los valores de gris de cada pixel del conjunto de imágenes utilizadas al realizar el PCA.

Los primeros vectores tendrán la información más importante del espacio, mientras que a medida que recorremos la matriz la información pasa a ser redundante. Por lo tanto, para formar el espacio de facciones no necesitaremos todos los vectores, lo que se traduce en que tendremos una reducción importante en la cantidad de información manejada. Como estos vectores representan la variación de los píxeles de las imágenes, y todas las imágenes empleadas son faciales, si representamos estos vectores como imágenes veremos que parecen “caras”, de ahí que también se les conozca como *eigenfaces*. La figura 5 es un ejemplo de representación de *eigenfaces*.

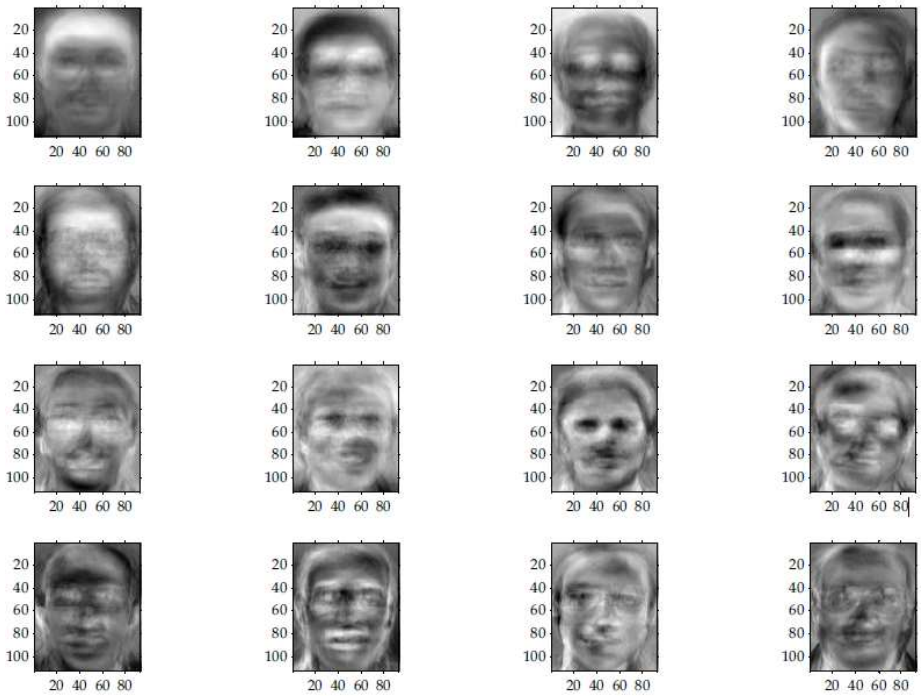


Figura 5. Representación de las 16 primeras eigenfaces

Para finalizar la fase de entrenamiento, las imágenes que se emplearon al realizar el PCA se proyectan en el espacio de facciones. La proyección caracteriza la imagen facial de un individuo como la suma de los diferentes pesos del espacio de imágenes.

En la fase de clasificación, una imagen facial desconocida es proyectada contra el espacio de facciones ya mencionado. Después, por medio de la distancia euclídea, se busca la imagen facial proyectada más parecida a la desconocida.

La técnica de PCA tienes dos defectos importantes. Al estar basado en los valores de brillo de las imágenes, es muy sensible a variaciones en la iluminación, por lo que es importante que para utilizar esta técnica la iluminación esté controlada. El otro defecto es que cuando se quieran añadir nuevos imágenes o sujetos a las que conformaban el entrenamiento original, hay que realizar de nuevo el PCA y volver a proyectar todas las imágenes de nuevo.

3.2.2 LDA

[3] Otro método común en el reconocimiento facial es el LDA (*Linear Discriminant Analysis* o Análisis Discriminante Lineal), también conocido como *Fisherfaces*.

En contraste con las *eigenfaces*, las *fisherfaces* buscan maximizar la varianza de las muestras entre clases (entre personas) y minimizarla entre muestras de la misma clase (de la misma persona). Con esto se logra obtener mejores resultados en caso de que haya variaciones de la iluminación y expresión respecto de las imágenes de entrenamiento. Para lograr esto, en la fase de entrenamiento se necesita tomar varias imágenes de cada sujeto en diferentes condiciones de iluminación y pose. De esta manera, la descomposición en *fisherfaces* interpola o extrapola las demás condiciones de

pose o iluminación que se pueden presentar aparte de las ya tomadas. La debilidad de la técnica está en que se necesitan varias tomas del mismo individuo y que estas sean representativas de las variaciones que se vayan a presentar en la aplicación real. Además, esta técnica es más costosa computacionalmente que la de PCA.

El proceso de reconocimiento es básicamente el mismo que en el PCA: primero, la imagen es proyectada al espacio formado por las imágenes de entrenamiento, para a continuación buscar la imagen más semejante por medio de la distancia euclídea. Lo que diferencia ambas técnicas es la forma de calcular dicho espacio.

Para calcular el espacio, se buscan dos matrices de covarianza, una inter-clase que corresponde a las diferentes imágenes de una misma persona, y una extra-clase que corresponde a las imágenes de diferentes personas. En la figura 6 podemos ver un ejemplo de dos formas distintas de proyectar los mismos puntos sobre una línea, una buscando minimizar la distancia entre puntos de una misma clase, y otra buscando maximizar la distancia entre puntos de distinta clase.

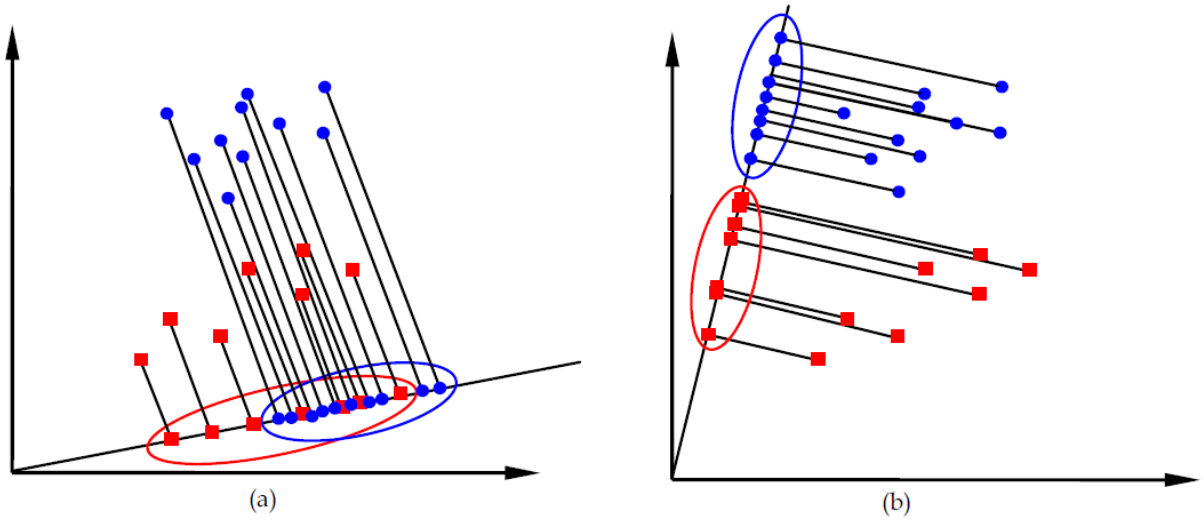


Figura 6 Puntos proyectados a una línea. (a) Minimizando la distancia (b) Maximizando la distancia

La relación entre ambas matrices nos darás el subespacio LDA, compuesto por los *eigenvectors* ya conocidos de la técnica PCA. Estos *eigenvectors* son las *fisherfaces* que dan nombre a la técnica. En la figura 7 puede verse un ejemplo de *fisherfaces*.

Figura 7 Representación de las 16 primeras Fisherfaces

3.2.3 Redes neuronales

[4] Tanto para la técnica PCA como para el LDA, en la parte final del proceso se emplea la distancia euclídea para encontrar la imagen más parecida a la que se está buscando. Para mejorar los resultados de ambas técnicas, dos arquitecturas nuevas son propuestas aplicando redes neuronales, la PCA-NN (*Neural Network*) y la LDA-NN.

El empleo de redes neuronales es similar en ambas técnicas. Después de calcular las *eigenfaces* (o las *fisherfaces* en caso del LDA), los vectores proyectados son empleados para entrenar la red neuronal. Cuando una nueva imagen se considera para el reconocimiento, es proyectada al espacio correspondiente (PCA o LDA), dando como resultado un vector. Este vector es aplicado a la red neuronal, y su salida es comparada a las de las imágenes de entrenamiento.

3.3 Técnicas basadas en imágenes 3D

El empleo de la información facial tridimensional permite mitigar el efecto negativo de algunos factores que son críticos en las técnicas anteriormente descritas, como la influencia de la iluminación

y la orientación del sujeto en la imagen. En las técnicas de reconocimiento en 2D, toda la información disponible se reduce a la intensidad de los pixeles, por lo que cualquier algoritmo siempre va a tener que afrontar dichos problemas. Para evitar esta dependencia, hay varios algoritmos propuestos basados en el modelado 3D de la cabeza. Para realizar este modelado, en algunas aproximaciones se han utilizado escáneres 3D [5]. En otros trabajos, el modelo 3D se crea a partir de múltiples imágenes del mismo sujeto tomadas desde un dispositivo 2D (ver figura 8 y 9) [6].

Al tener directamente el modelo 3D de la cabeza, se logra evitar la dependencia tanto de la iluminación como de la orientación de la cabeza respecto de la toma de la imagen. Aun así, la dependencia de la epresión del sujeto continúa existiendo.

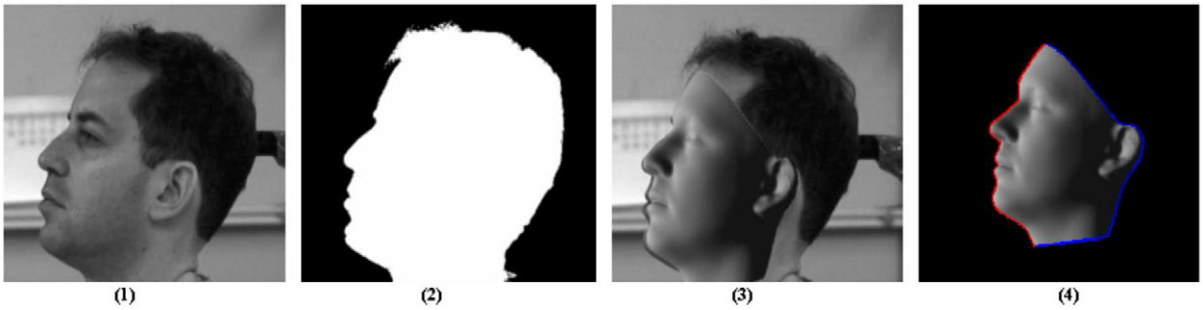


Figura 8 Extracción de silueta desde imagen 2D

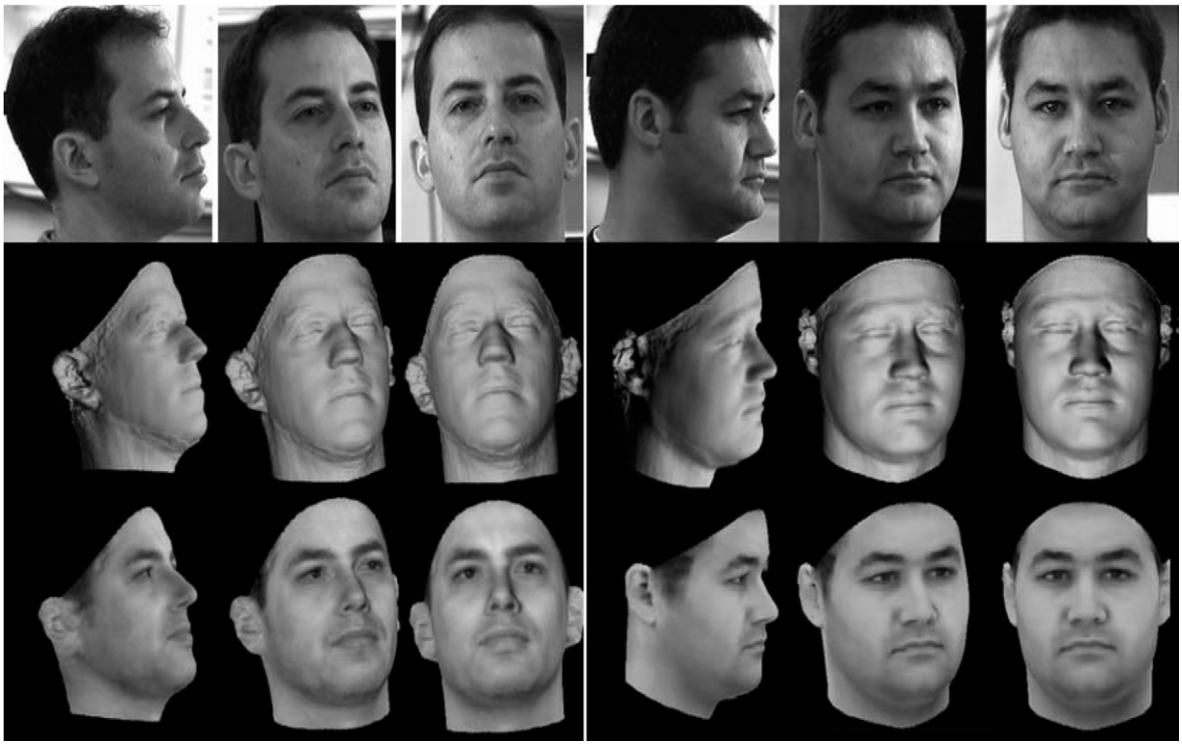


Figura 9 Modelos reconstruidos desde imágenes 2D

3.4 Alternativas

A pesar de que los principales esfuerzos en el reconocimiento facial se han puesto en las técnicas basadas tanto en imágenes fijas como en modelado 3D, hay muchas otras alternativas que han sido estudiadas.

En la solución propuesta por M. Tistarelli y E. Grosso [7] se utilizan dos cámaras para capturar imágenes "en estéreo" del sujeto a identificar, poniendo énfasis en detectar los ojos y la boca. Las imágenes son cortadas y remuestreadas para pasarlas a coordenadas polares. Por medio de operadores morfológicos, se busca extraer contornos, valles y picos en los valores de gris de la imagen, y así tener más rasgos característicos de cada individuo (figura 10).

Figura 10 Imagen original, contornos, valles y picos respectivamente

Otra solución interesante es la de V. E. Neagoe, A. D. Ropot y A. C. Mugioiu [8]. En ella se propone un algoritmo basado en redes neuronales mezclando imágenes del espectro visible con imágenes térmicas (figura 11). Este algoritmo se basa en la mejora que supone utilizar imágenes térmicas infrarrojas en situaciones en las que el control de la iluminación no es posible. Los sensores de infrarrojos miden la energía emitida por los cuerpos en lugar de la reflejada, por lo que el uso de estos sensores mejora los resultados en condiciones de pobre iluminación, o incluso en total oscuridad. Además, el reconocimiento con imágenes térmicas es más robusto frente a variaciones en la pose del sujeto. En este estudio se busca combinar las ventajas de ambos tipos de imágenes, evitando sus desventajas (figura 12).

Figura 11 Imágenes térmicas con variación de pose

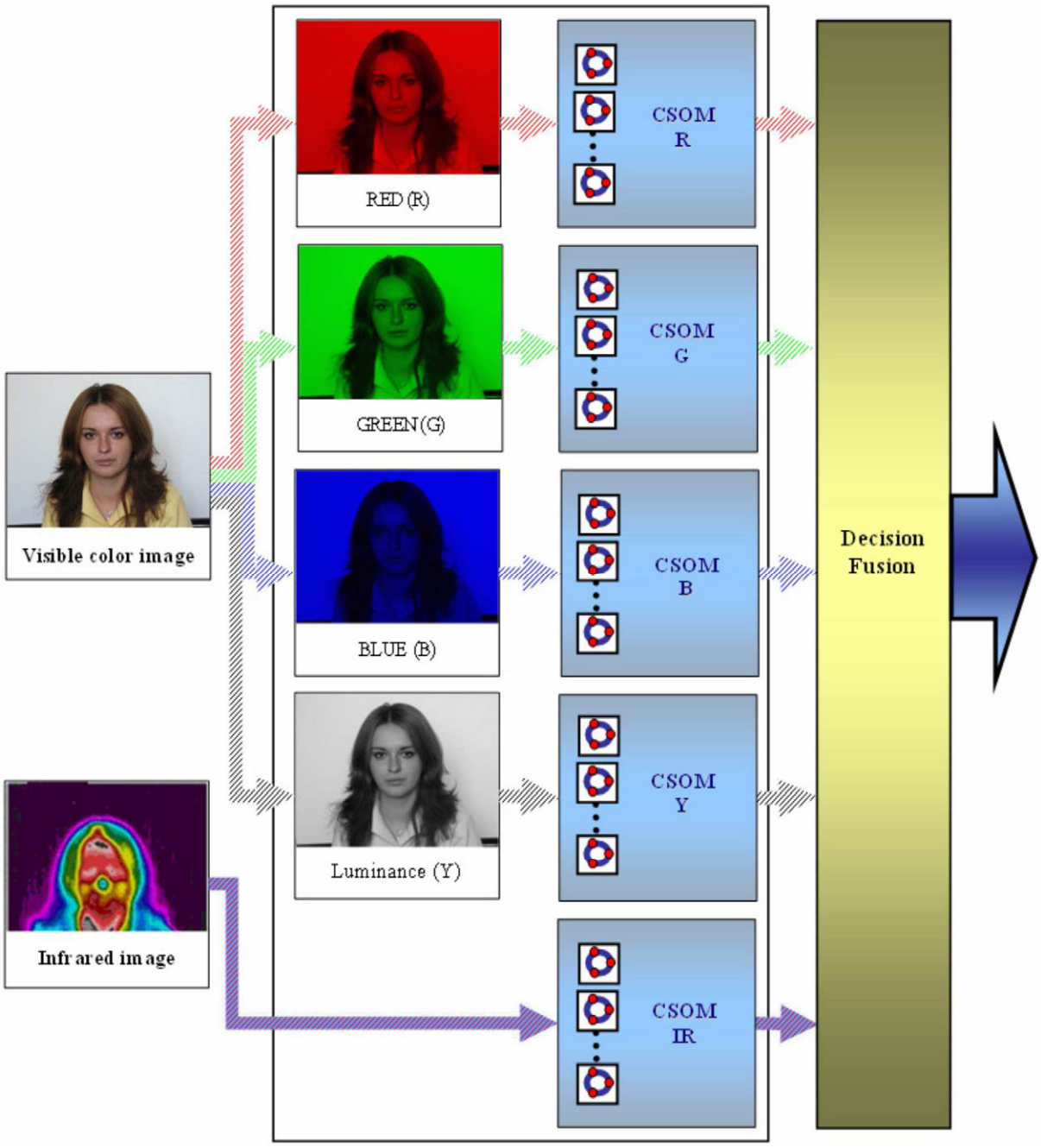


Figura 12 Reconocimiento basado en la mezcla de imágenes RGB con imágenes térmicas

Otro uso propuesto con imágenes térmicas es el de mejorar la técnica del PCA con el uso de estas en lugar de las imágenes convencionales [9]. Para ello, las imágenes térmicas son convertidas a coordenadas polares (figura 13). Esta conversión a coordenadas polares se realiza para evitar los problemas de rotación de la cabeza y la escala de la cara en relación a la imagen. Estas imágenes térmicas-polares son utilizadas para calcular las *eigenfaces* (figura 14). Finalmente, como clasificador se emplea un algoritmo *backpropagation* (propagación hacia atrás), uno de los más usados en el entrenamiento de redes nueronales.

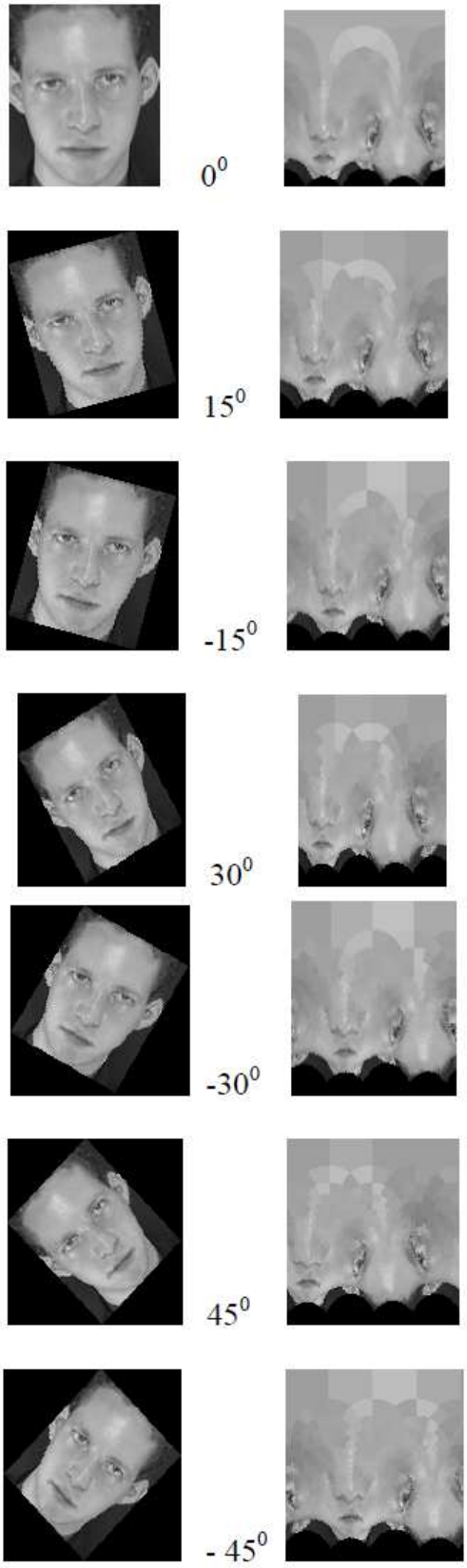


Figura 13 Imágenes con distintas orientaciones antes y después de la conversión a coordenadas polares

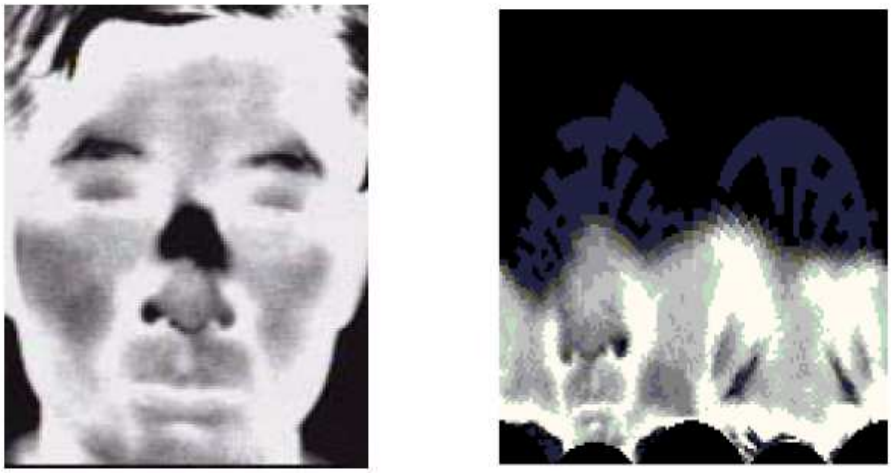


Figura 14 Imagen térmica y conversión a coordenadas polares

Una alternativa del reconocimiento con imágenes faciales es la de utilizar imágenes térmicas para extraer el sistema venoso y arterial superficial de la cara [10]. La manera de hacer esto es con la transformación *top-hat* blanca. Esta transformación extrae los detalles brillantes en presencia de sombras, lo que en una imagen térmica fácil coincide con los vasos sanguíneos cerca de la piel. Estas "huellas térmicas" extraídas de los vasos sanguíneos son diferentes para cada individuo, por lo que para realizar el reconocimiento se compara la posición, número y morfología de las bifurcaciones sanguíneas del sujeto a identificar con las de los almacenados en la base de datos (figura 15).

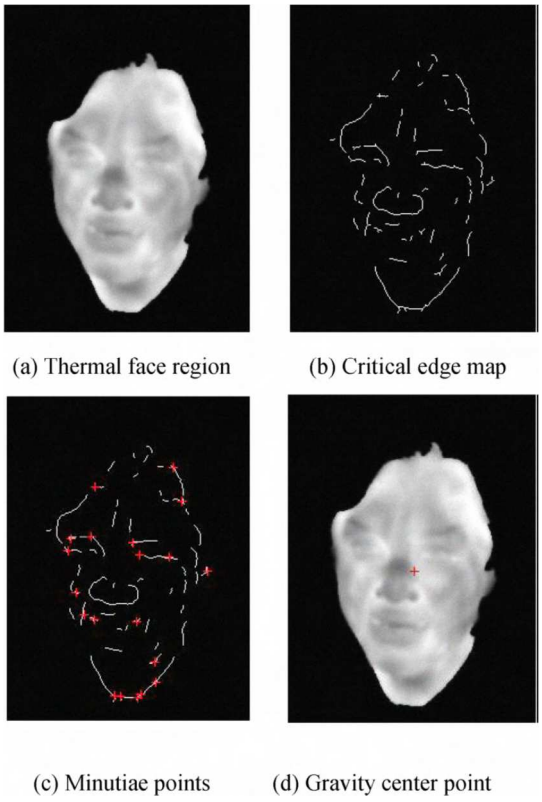


Figura 15 (a) Imagen térmica de la cara (b) Mapa de vasos (c) Bifurcaciones sanguíneas (d) Punto de centro de gravedad

4 SOLUCIÓN PROPUESTA

4.1 Descripción global del proyecto

En el presente proyecto se va a implementar una aplicación para reconocimiento facial en lenguaje C++. Para lograrlo, he utilizado de apoyo una de las bibliotecas libres más utilizadas en el entorno de procesado de imagen, Open CV. Esta biblioteca está desarrollada principalmente para C y C++. Actualmente se está desarrollando en otros lenguajes: Python, Matlab, Ruby, etc.

A la hora de plantear el algoritmo de la aplicación, he dividido el problema del reconocimiento facial en dos módulos diferentes e independientes entre sí:

- Adquisición de la imagen facial del sujeto a identificar y procesamiento de esta.
- Técnica de reconocimiento a utilizar: PCA

4.1.1 Adquisición de la imagen facial

La primera decisión importante de cara al funcionamiento de la aplicación es cómo será la toma de imagen (desde una imagen estática en condiciones controladas de iluminación, desde una secuencia de vídeo, etc.). Teniendo en mente una posible aplicación de control de acceso, se ha elegido una opción "mixta". La aplicación debe capturar continuamente el vídeo de entrada desde una webcam, buscando en los *frames* de la secuencia de vídeo una imagen facial. En caso de encontrar una, la aplicación procesa el *frame* capturado en ese momento y procede al reconocimiento facial.

Para la fase de entrenamiento, se han implementado dos maneras distintas de obtener las características faciales de los sujetos a identificar. Por un lado, está el método análogo al del reconocimiento, dónde un sujeto se situará delante de la cámara. La aplicación destacará la cara en la imagen, y pulsando una tecla la aplicación extraerá los datos faciales y los almacenará en su base de datos. La otra posibilidad, es preparar un archivo txt con el nombre de las personas, la ruta en el ordenador de las imágenes y el número de ellas, para que la aplicación se encargue de leerlas e ir una a una extrayendo las características y construyendo la base de datos. Este es el método que se ha utilizado para testear la capacidad de reconocimiento con bases de datos de imágenes de libre utilización.

4.1.2 PCA. Concepto

El análisis de componentes principales es un método estadístico de identificar patrones en un conjunto de datos y expresarlos de manera que se remarquen sus similitudes y sus diferencias. Dado

que dichos patrones son difíciles de identificar en datos de grandes dimensiones, el PCA es una herramienta muy útil para analizar datos.

El PCA construye una transformación lineal que escoge un nuevo sistema de coordenadas para el conjunto original de datos en el cual la varianza de mayor tamaño del conjunto de datos es capturada en el primer eje (llamado el Primer Componente Principal), la segunda varianza más grande es el segundo eje, y así sucesivamente.

La otra ventaja principal del PCA es que una vez encontrados dichos patrones, permite la compresión de los datos reduciendo el número de dimensiones, perdiendo poca información.

La forma de emplear el PCA en un conjunto de datos de dos dimensiones (*x*, *y*) es la siguiente:

1. Para que el PCA funcione correctamente, hay que restar la media de cada una de las dimensiones de los datos. La media restada es la media de cada dimensión. Así, a todos los valores *x* se les resta $\bar{x}$, y a todos los valores *y* se les resta $\bar{y}$, dando como resultado un conjunto de datos cuya media es 0 y al que llamaremos Datos Ajustados.
2. Se calcula la matriz de covarianza $\sigma(x, y)$ de Datos Ajustados.

$$cov(X,Y) = \frac{\sum_{i=1}^{n}(X_i - \overline{X})(Y_i - \overline{Y})}{(n-1)} \quad \textbf{(4)}$$

$$C = \begin{pmatrix} cov(x,x) & cov(x,y) \\ cov(y,x) & cov(y,y) \end{pmatrix} \quad \textbf{(5)}$$

3. Se calculan los *eigenvectors* (vectores propios) y *eigenvalues* (valores propios) de la matriz de covarianza. En algebra lineal, los *eigenvectors* de un operador lineal son los vectores no nulos que, cuando son transformados por el operador, dan lugar a un múltiplo escalar de sí mismos, con lo que no cambian su dirección. Este escalar recibe el nombre de *eigenvalue*. Cada *eigenvalue* está asociado a un *eigenvector*.

$$\begin{pmatrix} 2 & 3 \\ 2 & 1 \end{pmatrix} \times \begin{pmatrix} 3 \\ 2 \end{pmatrix} = \begin{pmatrix} 12 \\ 8 \end{pmatrix} = 4 \times \begin{pmatrix} 3 \\ 2 \end{pmatrix} \quad \textbf{(6)}$$

En este ejemplo, $\begin{pmatrix} 3 \\ 2 \end{pmatrix}$ sería el *eigenvector* y 4 su *eigenvalue* asociado.

4. El *eigenvector* con el *eigenvalue* más alto será el componente principal del conjunto de datos, por lo que habrá que ordenar estos vectores de mayor a menor *eigenvalue*. Esto hace que los componentes queden ordenados de mayor a menor importancia. Aquí es de dónde viene la capacidad de compresión del PCA, y es que se pueden descartar los *eigenvectors* menos importantes (los que tienen *eigenvalues* pequeños) perdiendo por ello muy poca información. A continuación, se forma una matriz con los *eigenvectors* en columnas, llamándose esta matriz *feature vector* (vector de características).

5. Por último, se crea el nuevo conjunto de datos. Para ello, se multiplica la traspuesta del vector de características a la traspuesta de la matriz de datos original, con el valor medio sustraído.

$$Datos\ Finales = Vector\ CaracterísticasT\ x\ Datos\ AjustadosT \quad \textbf{(7)}$$

6. Para reconstruir los datos originales, basta con multiplicar la inversa de Vector de Características traspuesto a los datos finales y sumarle el valor medio original. Si hemos quitado los *eigenvectors* menos importantes, se habrá perdido un poco de información.

$$Datos\ Originales = ((Vector\ CaracterísticasT)\text{-}1\ x\ Datos\ Finales) + Valor\ Medio \quad \textbf{(8)}$$

4.1.3 PCA en reconocimiento facial

Una imagen facial de tamaño NxN puede ser considerada como un vector de dimensión N^2, donde las filas de la imagen son colocadas una tras otra hasta formar una imagen de una sola dimensión (9). Los valores del vector son los valores de gris de la imagen.

$$X = (x_1 \quad x_2 \quad x_3 \ \cdots \ x_{N2}) \quad \textbf{(9)}$$

Para aplicar el PCA a las imágenes faciales de la fase de entrenamiento, primero se transforman las imágenes en vectores como se indica en el apartado anterior. A continuación, se forma una matriz con todas las imágenes, correspondiendo cada fila de la matriz a una imagen facial distinta (10). Esta matriz es a la que se aplicará el PCA, teniendo finalmente el conjunto total de las imágenes en términos de los nuevos ejes, sus *eigenvectors*.

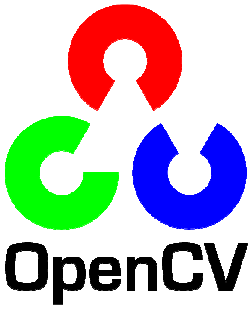
$$Matriz\ Imágenes = \begin{pmatrix} Vector\ Imagen\ 1 \\ Vector\ Imagen\ 2 \\ \vdots \\ Vector\ Imagen\ M \end{pmatrix} \quad \textbf{(10)}$$

Para realizar el reconocimiento, cada imagen de entrenamiento es proyectada a estos nuevos ejes. A continuación, la imagen facial de la persona a reconocer también es proyectada. Por medio de la distancia euclídea (11), se mide la diferencia valor por valor entre la imagen de entrenamiento proyectada y la imagen a reconocer. De esta forma, encontraremos la imagen más parecida a la que se ha de reconocer.

Para evitar falsos positivos, hay que definir un valor umbral a partir del cual la distancia euclídea sea demasiado grande como para considerar que la imagen a reconocer y la imagen de entrenamiento correspondan a la misma persona

$$d_E = \sum_{i=1}^{N2} \sqrt{(Valor\ Imagen\ Reconocimiento_i - Valor\ Imagen\ Entrenamiento_i)^2} \quad \textbf{(11)}$$

4.2 OpenCV



OpenCV es una biblioteca libre de visión artificial originalmente desarrollada por Intel. Desde que apareció su primera versión alfa en el mes de enero de 1999, se ha utilizado en infinidad de aplicaciones. Desde sistemas de seguridad con detección de movimiento, hasta aplicaciones de control de procesos donde se requiere reconocimiento de objetos. Esto se debe a que su publicación se da bajo licencia BSD (Berkeley Software Distribution o distribución de software berkeley), que permite que sea usada libremente para propósitos comerciales y de investigación con las condiciones en ella expresadas [11].

OpenCV tiene una estructura modular. Cada módulo incluye varias librerías enfocadas a un objetivo concreto. Los módulos principales son los siguientes:

- **core:** el módulo que define las estructuras básicas y funciones que serán usadas por otros módulos.
- **Imgproc:** dedicado al procesado de imagen, contiene funciones como transformadas lineares o no lineales, transformaciones geométricas, conversiones entre espacios de color, histogramas, etc.
- **Video:** módulo enfocado a funciones varias de vídeo, como seguimiento de movimiento, extracción del fondo y algoritmos de seguimiento de objetos.
- **Objdetect:** funciones de detección de objetos, incluyendo clases predefinidas (por ejemplo ojos, boca, coches, etc).
- **Highgui:** este módulo sirve para añadir un interfaz sencillo a las aplicaciones de imagen y vídeo (botones, barras de desplazamiento, etc).

Entre las múltiples ventajas que ofrece trabajar con una librería como OpenCV está la de tener clases propias y funciones preparadas para el procesado de imágenes y vídeo. Las utilizadas en el presente proyecto se detallaran en el apartado implementación.

5 DISEÑO

Una vez determinado el enfoque que se quiere dar a la aplicación, en este apartado se analizará, de manera independiente, cada una de las funciones que se desean poder realizar con la aplicación y cómo han sido implementadas:

- Captura y procesado de imagen facial.
- Entrenamiento.
- Reconocimiento.
- Test de reconocimiento con bases de datos de libre disposición.
- Mostrar eigenfaces.

5.1 Captura y procesado de imagen facial

La aplicación debe abrir la webcam del ordenador en el que se encuentre instalada, tomar imágenes, aplicar un clasificador para encontrar imágenes faciales y, una vez determinadas, recortar las imágenes y procesarlas para que todas sean en blanco y negro y tengan el mismo tamaño en píxeles. Además, se ecualizará el histograma de las imágenes para reducir el impacto de las variaciones de luz ambiental a la hora de capturar las imágenes.

Realizar todas estas funciones sería un trabajo costoso y laborioso si se partiera de cero, pero la librería OpenCV viene con funciones y clases ya preparadas para la mayoría de este tipo de necesidades.

Los diagramas serían los siguientes tanto para la lectura de imágenes en el entrenamiento cómo para el reconocimiento (figura 16). En el diagrama aparece una acción que es “Leer tecla”. Esta acción sólo aparece en la fase de entrenamiento, ya que se hace necesario la confirmación del usuario para que la imagen sea tomada como correcta (orientación, iluminación, evitar expresiones extrañas, etc.). Además, no es una acción de espera como tal, sino que la aplicación está continuamente leyendo el teclado mientras que la aplicación sigue leyendo imágenes y mostrándolas en pantalla hasta que el usuario confirme la toma de la imagen como buena. De esta forma, el usuario se puede colocar correctamente mientras que comprueba en el monitor su orientación respecto a la webcam y si ha sido encontrado por el clasificador.

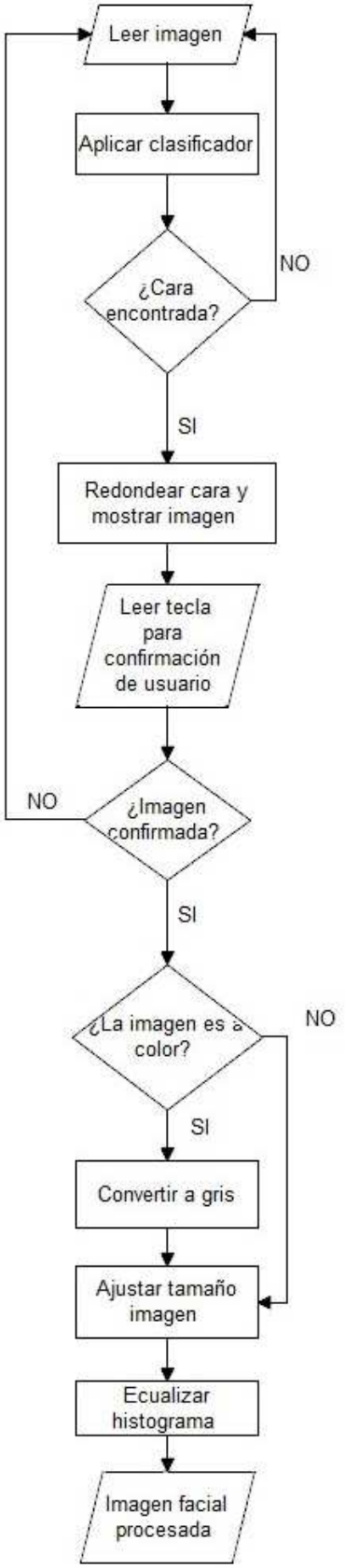


Figura 16 Diagrama lectura y procesado de imagen facial

Para definir el tamaño de las imágenes faciales recortadas se ha consultado en la ISO/IEC 19794 parte 5 [12]. La norma ISO/IEC 19794 define varios formatos de intercambio de datos biométricos, y en concreto la parte 5 se dedica en exclusiva a las imágenes faciales.

En esta norma se definen dos tipos de imágenes faciales, la básica y la frontal. A su vez, de la imagen frontal se definen otros dos tipos, *Full Frontal* o Frontal Completa y la *Token Frontal* o Frontal de Ficha. En las imágenes frontales completas se incluye el pelo, el cuello y los hombros (figura 17).

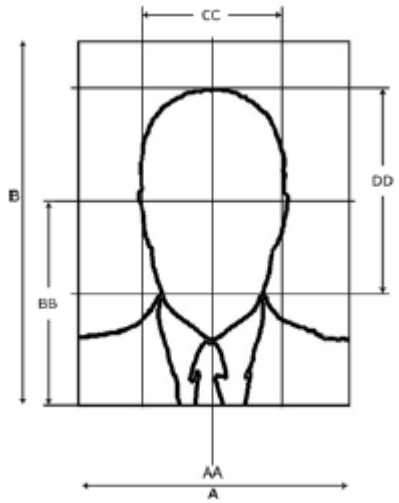


Figura 17 Imagen frontal completa

La imagen frontal de ficha se define según las especificaciones de la siguiente tabla:

Tabla 1 Especificaciones para imagen facial de ficha

Característica	Valor
Ancho de la imagen	W
Alto de la imagen	W/0.75
Coordenada Y de los ojos	0.6*W
Coordenada X del ojo derecho	(0.375*W)-1
Coordenada X del ojo izquierdo	(0,625*W)-1
Distancia entre ojos	0,25*W

Para la aplicación, lo que nos interesa es la proporción de ancho/alto de la imagen. El ancho de las imágenes recortadas se ha definido en 100 píxeles, por lo que el alto, de acuerdo a la tabla, será de 133. De esta forma, se tiene suficiente información para realizar el reconocimiento facial sin sobrecargar demasiado los procesos por utilizar imágenes mucho mayores, que además pueden dificultar el reconocimiento al tener mayor detalle y, por lo tanto, mayores variaciones en los valores de gris debido a las condiciones ambientales de luz. En la figura 18 se puede ver un ejemplo de imagen facial procesada por la aplicación

Figura 18 Imagen facial de 100x133 píxeles procesada por la aplicación

5.2 Entrenamiento

La fase de entrenamiento puede ser implementada para leer imágenes desde el disco duro, o directamente ir añadiéndolas desde la cámara del ordenador. Para el presente proyecto se ha elegido la segunda opción, ya que el tomar las imágenes desde la cámara del ordenador implica que las estemos tomando de manera muy similar a cómo tomaremos las imágenes para el reconocimiento (condiciones ambientales, resolución, sombras), mientras que si las imágenes son leídas desde un archivo, estas, al estar tomadas de manera arbitraria, dificultarán el reconocimiento.

También hay que tener en cuenta el algoritmo del reconocimiento facial con PCA. El PCA utiliza el conjunto global de las imágenes de entrenamiento para obtener la matriz de covarianza y los *eigenvectors*. Esto implica que, inevitablemente, cada vez que se añada una nueva persona a la base datos, incluso cuando se añada una nueva foto de un sujeto ya existente, habrá que volver a realizar el PCA a todo el conjunto global de imágenes de entrenamiento. En bases de datos de pocos usuarios estos procesos serán transparentes, pero a medida que estás aumentan puede llegar a ser problemático. De ahí la elección de imágenes de un tamaño moderado (100x133) para no añadir más carga a los procesos.

Igualmente, este problema sólo se da en la fase de entrenamiento. En la fase de reconocimiento bastará con realizar una vez el PCA al conjunto de las imágenes y a continuación se podrá seguir trabajando con los *eigenvectors* obtenidos.

Para finalizar la fase de entrenamiento, una vez realizado el PCA se transforman una a una todas las imágenes faciales de entrenamiento utilizando los nuevos ejes, los *eigenvectors*. Estas imágenes transformadas serán las que se comparen con imágenes de caras desconocidas.

En resumen, a la hora de añadir una nueva imagen facial, habrá que añadirla al conjunto total de imágenes, recalcular los *eigenvectors* y volver a transformar una a una las imágenes de entrenamiento.

5.3 Reconocimiento

A la hora de realizar el reconocimiento de un sujeto frente a la cámara, primero tendremos que realizar el procesado de la imagen como se ha visto en el apartado 5.1 y a continuación, se proyectará dicha imagen es los *eigenvectors* calculados en el punto 5.2. Por último, con la imagen

proyectada, se busca cuál es la imagen facial de entrenamiento más parecida, y si es inferior al valor umbral, será dado por bueno el reconocimiento.

Para el diagrama (figura 19), partimos de una imagen facial ya procesada y lista para su reconocimiento.

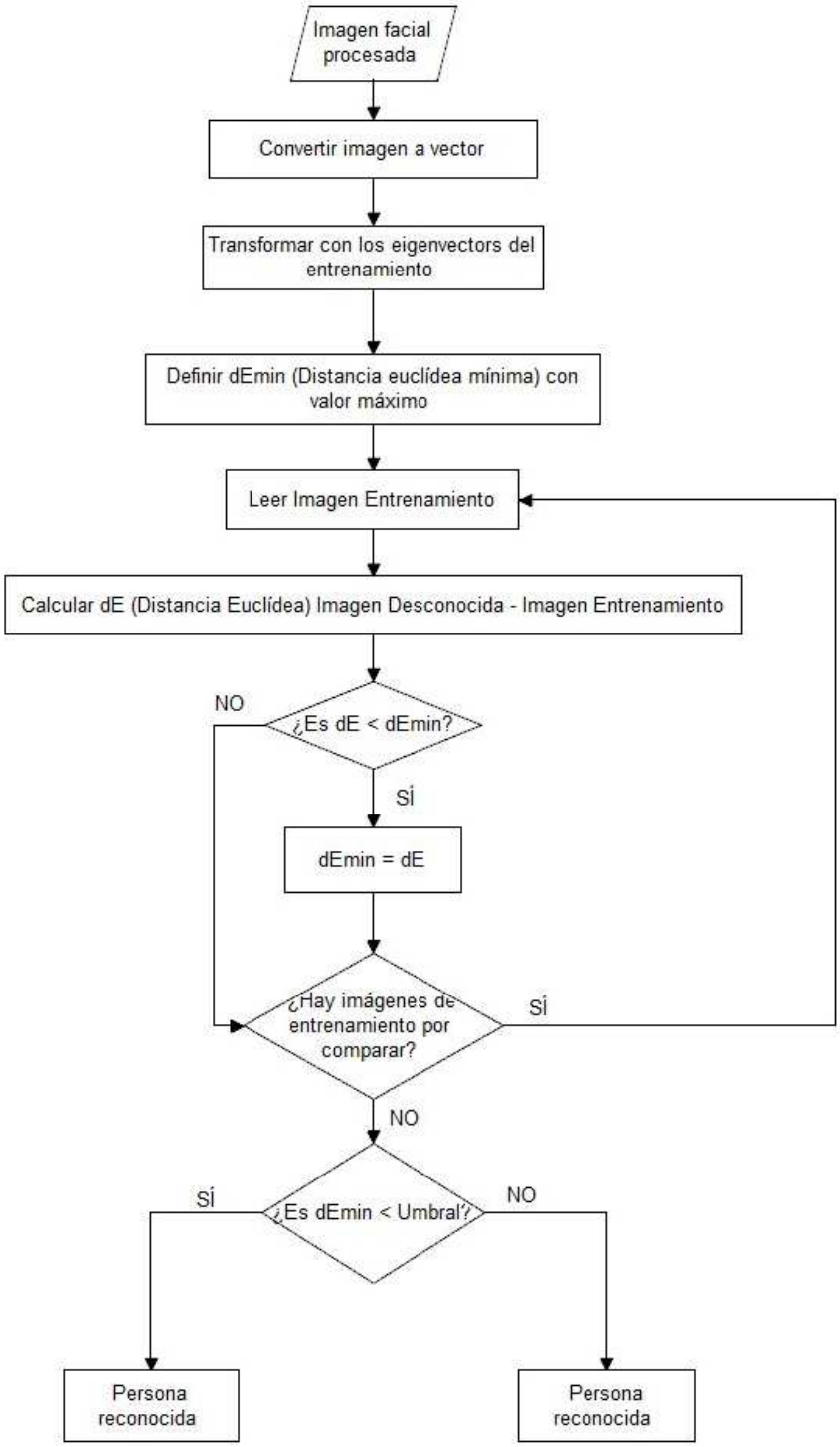


Figura 19 Diagrama reconocimiento

5.4 Test de reconocimiento

En la fase de reconocimiento, se ha presentado un valor umbral. Este valor umbral es necesario debido a que el algoritmo del reconocimiento con *eigenfaces* finaliza al buscar la cara más parecida entres las de la fase de entrenamiento. Pero esto no implica que las imágenes ni siquiera sean similares entre sí, sólo indica cuál es la más similar entres las ya almacenadas. Para controlar esta situación se ha de introducir un valor umbral.

En el momento del reconocimiento, este valor umbral se compara con el valor de la mínima distancia euclídea obtenida. Si el umbral es inferior, quiere decir que el reconocimiento nos ha dado un falso positivo, lo que significa que la cara no ha sido reconocida en la base de datos. Pero este valor umbral no puede ser demasiado bajo, ya que de lo contrario vamos a descartar muchos reconocimientos que realmente son positivos. El valor umbral debe ser un valor que nos dé una relación aceptable entre positivos y falsos positivos.

Para definir el valor umbral, se realizarán varias pruebas con una base de datos de imágenes faciales de libre utilización. Estas bases de datos tienen varias imágenes faciales del mismo sujeto, por lo que extraeremos una de ellas y aplicaremos al resto el PCA. Finalmente, aplicaremos el reconocimiento a la foto extraída. Como conocemos la identidad del sujeto, conoceremos de antemano si el resultado es positivo o falso positivo.

Este procedimiento se repite varias veces con distintos valores de umbral hasta hallar el más adecuado.

5.5 Mostrar *eigenfaces*

La última parte de la aplicación no tiene utilidad práctica de cara a realizar el reconocimiento, pero sí es interesante a nivel informativo, y es la posibilidad de representar los primeros *eigenvectors* del conjunto de imágenes de entrenamiento, las llamadas *eigenfaces*.

6 IMPLEMENTACIÓN

En la fase de implementación, lo primero es elegir el entorno de trabajo. Teniendo en cuenta que la aplicación se desarrolla en un ordenador con Windows 7, y la elección de OpenCV como librería de procesado de imagen, se optó por Microsoft Visual Studio como IDE (*Integrated Development Environment* o Entorno de Desarrollo Integrado). La decisión fue tomada debido a la gran cantidad de información que hay disponible en la propia página del proyecto [13], así como en numerosas páginas de colaboración de programadores [14].

La versión utilizada en el presente proyecto es Microsoft Visual Studio 2010 Professional. Desde la propia página de Microsoft hay una versión gratuita disponible para estudiantes.

Para estudiar cómo se ha realizado la implementación de la aplicación, dividiremos de nuevo la aplicación en las dos partes principales del mismo: captura y procesado de la imagen facial y reconocimiento. Pero antes, se detallaran algunas clases propias de OpenCV que serán útiles en ambas partes.

6.1 Clases propias de OpenCV

A continuación se detallan algunas clases y funciones de la librería OpenCV que, de forma general, se utilizan en el proyecto:

- Clase *Mat*: es una estructura básica de OpenCV. Representa una matriz de datos de N dimensiones, pudiendo ser monocanal o multicanal. En la aplicación se utilizará principalmente para almacenar imágenes en color o en blanco y negro, pero también puede ser utilizada para almacenar vectores de valores reales o complejos, matrices, histogramas, etc. Las funciones de esta clase que se han utilizado son:
  - *push_back*: Añade nuevos elementos al final de la matriz.
  - *size*: Devuelve el tamaño de la matriz.

- *empty*: Indica si el objeto está vacío.
- *rows / cols*: Devuelven el número total de filas o de columnas, y permiten acceder a posiciones concretas de la matriz.
- *reshape*: Cambia la forma de la matriz.
- *convertTo*: Sirve para convertir el tipo de datos de una matriz a otro. Además permite escalar los datos.
- *release*: Sirve para anular la matriz y liberar memoria manualmente.

o Clase *Rect*: Almacenada las coordenadas de un rectángulo.

o Ventanas de OpenCV: Al ser una librería enfocada a la visión artificial y el procesado de imágenes y vídeo, OpenCV incluye unas sencillas ventanas para representar los resultados de las operaciones. Las funciones que utilizan son:

- *namedWindow*: Crea una ventana.
- *Imshow*: Sirve para mostrar imágenes en una ventana abierta. Si se utiliza en un bucle, permite la reproducción de vídeo.
- *destroyWindow*: Cierra una ventana abierta.

o función waitKey(): esta función es muy importante en OpenCV. Sirve para que finalicen todos los procesos que hubiera pendientes en la aplicación. Se puede usar de dos formas: con waitKey(0) se espera a que el usuario pulse una tecla para reanudar el programa, mientras que si sustituimos el 0 por un valor entero, ese valor será el tiempo en milisegundos que la aplicación esperará a que el usuario pulse una tecla. Si en ese tiempo no se ha pulsado nada, la aplicación continuará con la ejecución del código.

6.2 Trabajos anteriores

La implementación del reconocimiento mediante *eigenfaces* del presente proyecto está basada en la solución propuesta por Robin Hewitt para la revista SERVO Magazine en Abril del 2007 [15] y Mayo del 2007 [16]. Dicho artículo solo se refiere al entrenamiento y reconocimiento de imágenes desde el disco duro. Además, está desarrollada con una versión antigua de OpenCV que utiliza clases y funciones obsoletas en lenguaje C.

Otra solución propuesta que ha servido de apoyo ha sido la de Shervin Emami en su página web [17]. Esta solución está basado en la de SERVO Magazine, solo que añadiendo la implementación de la detección de una cara desde la webcam para compararla con las almacenadas en el disco duro. Igual que el artículo de SERVO Magazine, está desarrollada en una versión antigua de OpenCV basada en C.

Para la aplicación del presente proyecto, se ha tomado el algoritmo de reconocimiento con PCA de Servo Magazine y se ha modificado para utilizar las nuevas clases de OpenCV en C++. De la implementación de Shervin Emami se ha modificado la toma de imágenes faciales desde webcam de forma que también se utilice en el entrenamiento. Además, también se ha actualizado el código con las nuevas clases de C++, y se han añadido algunas decisiones de diseño, cómo la espera de confirmación por parte del usuario de las imágenes como válidas, la introducción del nombre del usuario, etc.

Para el resto de la aplicación se han definidos clases y funciones propias para gestionar las imágenes del disco duro, así como se ha estructurado el código de manera modular y siguiendo el principio de encapsulamiento.

6.3 Implementación captura imagen facial

Implementar esta parte de la aplicación es sencilla gracias a la librería OpenCV. Con la clase *VideoCapture* podremos abrir la cámara por defecto del sistema operativo y capturar los *frames* de vídeo en objetos de la clase *Mat*. Estos *frames* pueden ser entonces tratados como si fueran imágenes. Es a estas imágenes a las que aplicaremos el clasificador para encontrar las caras. En la figura 20 puede verse un ejemplo de una imagen tomada desde la webcam en la que se ha detectado una cara.

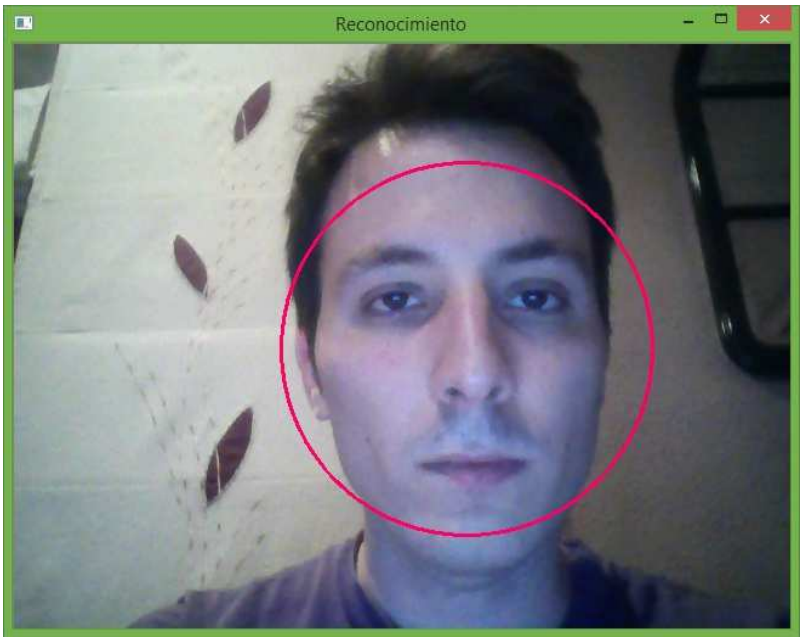


Figura 20 Ejemplo de detección facial

Para independizar el funcionamiento de la aplicación de cómo se realice la detección de la cara, se ha separado el código en los ficheros funcimagen.h y funcimagen.cpp. En ellos se han definido las funciones EncontrarCara, CirculoCara y ProcesadoCara.

- EncontrarCara: A esta función se le pasa un *frame* y los valores mínimos de anchura y altura de la cara en píxeles que debe tener la cara a encontrar. Primero se convierte la imagen a escala de grises y a continuación se le aplica un clasificador. El clasificador ya está programado en la librería OpenCV con la clase *CascadeClassifier*. Para utilizarlo, primero hay que crear un objeto de tipo *CascadeClassifier*:

  ```
  CascadeClassifier clasificador_facial;
  ```

  A continuación, abrimos el clasificador con el nombre que previamente se ha definido en un string constante dentro del archivo funcimagen.cpp. A la vez que se abre se comprueba que

el archivo exista para que en caso de no existir el programa se finalice y no se pueda quedar bloqueado en un bucle.

```
if( !clasificador_facial.load( archivo_clasificador ) ){
       printf("--(!)Error al abrir el clasificador facial. Pulse una tecla para
       salir\n");
       waitKey();
       exit(1);
}
```

Para aplicar el clasificador a una imagen, se utiliza la siguiente línea:

```
clasificador_facial.detectMultiScale( frame, caras, 1.1, 3,
0|CV_HAAR_FIND_BIGGEST_OBJECT, Size(ancho_min_cara, alto_min_cara) );
```

Los parámetros empleados sirven para definir el tamaño mínimo de la cara a encontrar y que el clasificador sólo se quede con la cara encontrada más grande.
Por último, la función devuelve la posición de la cara en un objeto de la clase *Rect*.

- CirculoCara: Esta función se utiliza para dibujar un círculo alrededor de una cara en una imagen pasada. Para ello, llama a la función EncontrarCara, y una vez tenga la ubicación definida, dibuja un círculo con la función de OpenCv *ellipse*.
- ProcesadoCara: Similar a la función anterior, solo que en lugar de dibujar un círculo, prepara la imagen para aplicarla el PCA, es decir, la pasa a escala de grises, la escala al tamaño definido en las constantes y ecualiza su histograma. El escalado de la imagen y la ecualización del histograma se realizan con las funciones de OpenCV *resize* y *equalizeHist*. Como se comentó en el apartado de desarrollo, para realizar el escalado se comprueba la relación de aspecto de la cara con la definida por las constantes (100x133). En la figura 21 se puede observar un ejemplo de la imagen que devuelve esta función.

Figura 21 Imagen facial procesada

La clase *CascadeClassifier* es la implementación del algoritmo de Viola y Jones que se detalló en el apartado de desarrollo. Además, la librería de OpenCV incluye varios clasificadores en formato xml ya entrenados para reconocer imágenes faciales:

- **haarcascade_frontalface_default.xml**
- **haarcascade_frontalface_alt.xml**
- **haarcascade_frontalface_alt_tree.xml**
- **haarcascade_frontalface_alt2.xml**

Tras varias pruebas, se descarta el primer clasificador al dar problemas detectando objetos como caras (figura 22). Al no detectar diferencias apreciables entre el resto de ellos, el clasificador que se utilizará en el proyecto será el de **haarcascade_frontalface_alt.xml**.

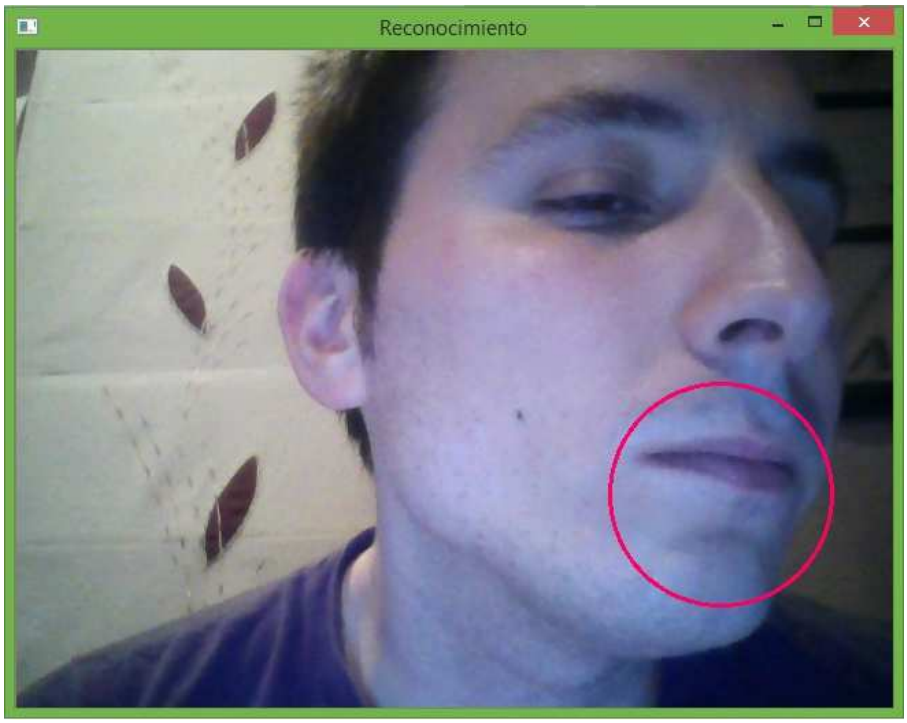


Figura 22 Ejemplo de detección facial errónea

6.4 Gestión de imágenes y usuarios

Para almacenar la información de los usuarios y sus imágenes se ha definido un archivo txt llamado imgref.txt que contiene las rutas de dichas imágenes y los datos de la persona asociada. Al iniciar la aplicación, se leerá dicho txt para preparar las variables que necesitará la aplicación. Una vez que se han preparado las variables, sólo habrá que acceder al txt en caso de querer añadir una nueva persona o una foto.

El fichero está compuesto por un conjunto de líneas, correspondiendo cada una de ellas a una imagen facial (figura 23). Las líneas tienen el siguiente formato:

Referencia	Archivo imagen	Estado	Nombre completo

- La referencia es el identificador único que asigna la aplicación a cada usuario.
- Archivo imagen es el nombre del archivo que corresponde a la fila. Está compuesto por la ruta (la subcarpeta *imagenes*) y el nombre del archivo jpg, que a su vez está compuesto por la referencia seguida de un guion bajo y el número de foto del mismo usuario.
- Estado es la letra A de usuario Activo o B de usuario Borrado. La idea está tomada de las bases de datos relacionales, en las que en general no se borran los registros sino que se quedan almacenados pero etiquetados como borrados o no activos. De esta forma, la

numeración de las referencias nuevas es consecutiva y no hay saltos en el orden de estas, aunque luego al realizar el PCA no se incluyan los usuarios borrados.

- Nombre completo es el nombre de la persona a la que corresponde la imagen facial.

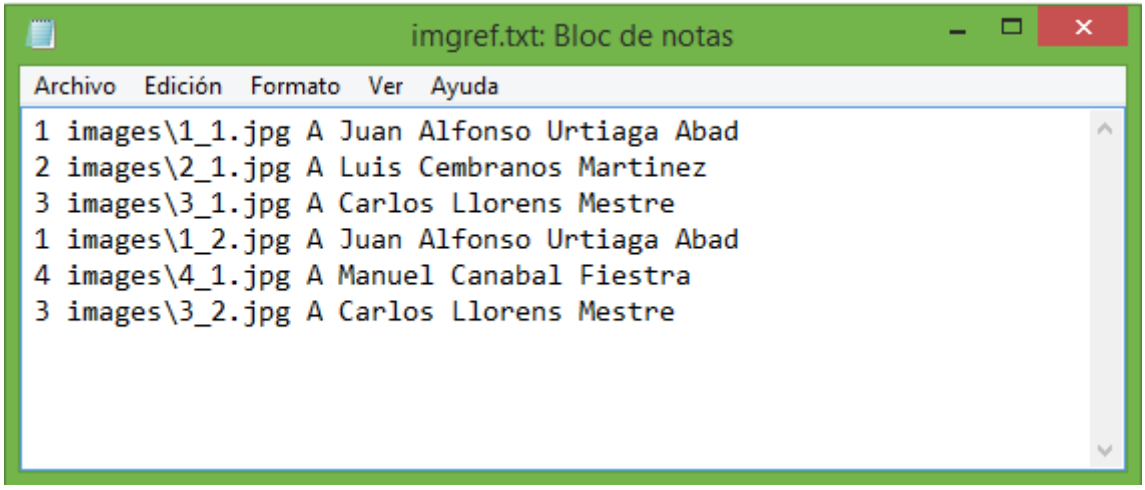


Figura 23 Ejemplo de contenido de imgref.txt

Al cargar la aplicación, esta lee una a una las líneas del archivo txt y almacena todos estos datos en un objeto. A este archivo sólo se volverá a acceder en caso de querer realizar alguna modificación (añadir una persona nueva, modificar una foto, etc.).

Para lectura y escritura del archivo se utiliza la librería estándar *fstream*, en concreto las clases *ifstream* y *ofstream*. La clase *ifstream* se utiliza para abrir el archivo en modo lectura y recorrerlo línea por línea. La clase *ofstream* abre el archivo en modo escritura, de forma que se puedan añadir líneas al final del archivo.

Una alternativa que se valoró en la fase de diseño fue la de crear una base de datos de SQL que se gestionara desde la propia aplicación, pero esto se escapa del alcance del presente proyecto.

6.5 Implementación reconocimiento

Para la implementación de las funciones y clases del reconocimiento, se han creado dos ficheros funcreconocimiento.h y funcreconocimiento.cpp dónde se implementan las clases y funciones necesarias para esta parte de la aplicación.

A continuación se detallan algunas clases y funciones propias de OpenCv utilizadas:

- imread: Lee una imagen en el archivo especificado y la guarda en un objeto *Mat*.
- imwrite: Guarda una imagen en el disco duro en un archivo formato jpg.
- Clase PCA: La clase PCA es la implementación del cálculo de los *eigenvectors* y los *eigenvalues* en OpenCV. Esta clase viene programada para recibir como entrada una matriz de datos. De esta matriz calcula la matriz de covarianza, y a continuación calcula los *eigenvectors*. También podemos elegir cuántos *eigenvectors* queremos mantener para así descartar los de menor importancia. En la aplicación nos quedaremos con tantos vectores

como el número total de imágenes de entrenamiento. A medida que aumente el número de imágenes de entrenamiento, los *eigenvectors* cada vez tienen menos importancia, y el mantenerlos lo único que genera es mayor tiempo de computación, por lo que se impone un límite de 100 *eigenvectors*. Todo esto se hace con una sola línea de código de la siguiente manera:

```
pca(datos,noArray(),CV_PCA_DATA_AS_COL,total_eigenvectors);
```

En esta línea de código, *datos* es el conjunto de imágenes faciales de la base de datos transformadas en columnas.

Adicionalmente, la clase PCA incluye dos funciones muy útiles para el presente proyecto.

- project: Esta función es muy importante en la programación del reconocimiento facial. Es la función que permite cambiar los ejes de una imagen y ponerla en función de los *eigenvectors*. Estas imágenes proyectadas serán las que se comparen con la distancia euclídea para realizar el reconocimiento facial.
- eigenvectors: con esta función podemos acceder a los *eigenvectors*.

Para implementar todas las variables y funciones que se necesitan para el reconocimiento, se ha definido la clase BDCaras (figura 24).

BDCaras
- v_caras : vector <Mat> - refs : vector <int> - v_nombres : vector <Mat> - v_caras_pro : vector <Mat> - ultima_ref : int - alto_caras : int - ancho_caras : int - total_eigenvectors : int - pca : PCA - ruta : String - String nombre_archivo_img : String
+ BDCaras () + nueva () + carga_datos (String : s_ruta , String : s_archivo_img) : int - carga_datos_test (String : s_ruta , String : s_archivo_img) : int + mostrar_eigenfaces () + agregar_persona (Mat : cara, String : s_nombre) : int + agregar_foto (Mat : cara , int : iRef) : bool + borrar_persona (int : iRef) : bool + buscar_cara (Mat : cara) : int + buscar_cara (Mat : cara , String s_nombre) : int + test_reconocimiento (int : iTotal , int : iPositivo , int : iFalsoPositivo) : int + buscar_persona (String s_nombre_persona) : int + total_imagenes_usuario (int iRef) : int - aplicar_pca ()

Figura 24 Clase BDCaras

En la clase BDCaras se han implementado todas las variables y funciones para gestionar el reconocimiento facial, desde leer y administrar las imágenes almacenadas en el ordenador hasta el reconocimiento de una cara en tiempo real.

Cómo son varios los datos que se tienen que gestionar, y cada vez que se quiere hacer algún proceso con ellos hay que realizar varios pasos previos, se han declarado todas las variables privadas. De esta forma, sólo se pueden acceder a estas desde las funciones públicas.

En el anexo B se encuentran los ficheros de cabecera donde se pueden ver las declaraciones de la BDCaras con comentarios sobre su utilización.

6.5.1 Clase BDCaras. Atributos

Los atributos de la clase BDCaras son los siguientes:

- v_caras: Es el vector dónde se guardan las imágenes faciales una vez leídas desde el disco duro. De esta manera, si hay que volver a aplicar el PCA no hará falta volverlas a leer.
- refs: Es el vector que almacena las referencias de las imágenes faciales. Las referencias son los identificadores que se le asignan a cada uno de los sujetos. Cada sujeto tendrá una referencia única en la base de datos. Están ordenados de la misma forma que el vector v_caras, de manera que para la misma posición *i*, en uno de los vectores tendremos la imagen facial de un sujeto y en el otro vector su referencia.
- v_nombres: al igual que el vector refs, este vector contiene los nombres de los sujetos en la base de datos.
- v_caras_pro: Este vector es análogo al de v_caras, ya que mientras que en v_caras se almacenan las imágenes faciales de los sujetos, en v_caras_pro se almacenan las mismas imágenes proyectadas en los *eigenvectors*. Este es el vector que se recorre a la hora de realizar el reconocimiento, y tiene el mismo orden que los vectores v_caras y refs.
- ultima_ref: Este campo indica la última referencia usada. De esta forma, a la hora de añadir un nuevo sujeto, se mira este campo para saber cuál es la próxima referencia sin usar. Así se evita duplicar referencias para sujetos distintos.
- alto_caras y ancho_caras: estos campos reflejan las dimensiones de las imágenes faciales con las que se está trabajando. Estas dimensiones están definidas como constantes globales en la cabecera del proyecto ya que afectan al procesado que se realiza en la detección de imagen. La clase BDCaras no define las dimensiones ya que las imágenes faciales se procesan en un módulo aparte de la aplicación, solo almacena su valor.
- total_Eigenvectors: sirve para determinar el número de *eigenvectors* que mantendremos al realizar el PCA. En general, nos quedaremos el número de imágenes faciales que se han aplicado al PCA, pero se define una constante como límite para no mantener un número excesivo de *eigenvectors*, ya que aumentaría mucho el procesado de los cálculos sin aportar apenas información.
- pca: es el objeto que aplica el PCA a las imágenes faciales, almacena los *eigenvectors* y proyecta en ellos las imágenes de entrenamiento y las del reconocimiento.

- ruta: almacena el nombre de la subcarpeta de la aplicación en la que se almacenan las imágenes de los sujetos. El nombre de la carpeta está definida como constante global del proyecto, por lo que es un parámetro que le pasaremos al objeto al inicializarlo.
- nombre_archivo_img: es el nombre del archivo txt donde están almacenadas las rutas de las imágenes, sus referencias y los nombres de los usuarios asociados. Igual que el nombre de la carpeta, el nombre del archivo está definido como constante global del proyecto.

6.5.2 Clase BDCaras. Operaciones

Para cumplir con el principio de encapsulamiento, todos los atributos son declarados privados. De esta forma, se logran dos objetivos: por un lado, que todas las acciones relacionadas con el reconocimiento estén manejadas internamente por la clase en vez de tener que implementar el código en el archivo principal. Por otro, minimizar el impacto de un cambio de técnica de reconocimiento en el algoritmo. Si esto sucediera, la implementación del fichero main.h apenas necesitaría ser modificada, ya que las declaraciones de las funciones disponibles en la clase BDCaras son genéricas e independientes de la técnica implementada. Las funciones son las siguientes:

- BDCaras / nueva: BDCaras es el constructor de clase. Este llama a la función *nueva*, que deja los atributos vacíos y listos para realizar la carga de datos.
- carga_datos: la función *carga_datos* recibe tanto la ruta de la carpeta en la que se almacenan las imágenes como el archivo txt que contiene la información de estas. Para hacerlo, primero abre el archivo imgref.txt y lo recorre línea por línea. Cada vez que lee una línea, y siempre que el usuario esté marcado como activo, va añadiendo las imágenes faciales, las referencias y los nombres a sus respectivos vectores. A la vez, va comparando las referencias para tener en cuenta cuál es la más alta. Por último, y siempre que haya cargado más de un usuario distinto, llama a la función *aplicar_pca*, que será la que prepare la aplicación para el reconocimiento.
- carga_datos_test: esta función se utiliza para realizar test con bases de datos de imágenes faciales. Es idéntica a la anterior con una modificación. En las pruebas con bases de datos hay que tener en cuenta al cargar las imágenes almacenadas en el disco duros que estas no están procesadas, por lo que hay que aplicar el detector de caras y el procesado antes de aplicar el PCA.
- aplicar_pca: esta función es privada ya que solo necesita ser llamada cuando haya algún cambio en las imágenes faciales de entrenamiento (p.e. cuando se carguen los datos al inicio de la aplicación, se añada o modifique una foto, etc.), por lo que no es necesario llamarla desde fuera de la clase. Además, no es necesario pasarle ningún parámetro ya que toda la información que necesita está disponible en el objeto BDCaras. Lo primero que hace esta función es transformar todas las imágenes almacenadas en el vector *v_caras* en columnas. Para ello, se crea un objeto *Mat* auxiliar donde cada una de las columnas será una imagen facial con sus valores de gris ordenados de forma consecutiva. El código de la transformación de las imágenes en columnas es el siguiente

```
Mat datos(total,v_caras.size(),CV_32FC1);
for (int i = 0; i < v_caras.size(); i++)
{
```

```
        Mat aux = datos.col(i);
        v_caras[i] = v_caras[i].clone();
        v_caras[i].reshape(1,total).col(0).convertTo(aux,CV_32FC1, 1/255.);
}
```

En este bucle, *datos* es la matriz donde se van a almacenar las imágenes en columnas. Se declara indicando el número de filas que tendrá (tantas como píxeles tienen las imágenes), el número de columnas (tantas como imágenes faciales vayamos a usar) y el tipo de datos a usar. A continuación, con el bucle *for* se transforman una a una todas las imágenes. El crear una matriz *aux* que apunte a la columna de salida dentro del bucle *for* es por la forma que tiene OpenCV de reservar la memoria para los objetos Mat. La línea de código en la que se clona una imagen de una cara a sí misma es porque para utilizar la función *reshape* las matrices deben ser continuas, y de esta forma la matriz v_caras[i] pasa a serlo (recordamos que v_caras es el vector de matrices dónde se almacenan todas las imágenes faciales).

Es a esta matriz *datos* a la que se le aplica el PCA para calcular los *eigenvectors*. Por último, queda proyectar las imágenes faciales en los nuevos ejes, pero no las imágenes faciales de *v_caras*, sino las transformadas en columnas y almacenadas en la matriz auxiliar. Cada columna de esta matriz se proyecta en los *eigenvector* y se añade al vector de imágenes proyectadas, *v_caras_pro*. De esta forma, en la memoria asignada a la aplicación tendremos las imágenes faciales de entrenamiento ya listas para compararse con una imagen facial desconocida, y solo será necesario volver a llamar a esta función en caso de añadir o modificar alguna imagen facial. El código de la proyección de las imágenes y el almacenamiento en el objeto es el siguiente:

```
for (int i = 0; i < v_caras.size(); i++)
{
        Mat cara_proyectada(total_eigenvectors,1,CV_32FC1);
        pca.project(datos.col(i), cara_proyectada);
        v_caras_pro.push_back(cara_proyectada);
}
```

- agregar_persona: como entrada recibe una imagen facial ya procesada y el nombre completo de la persona. La imagen se añade al vector *v_caras*, mientras que el nombre se añade al vector *v_nombres*. A continuación se le asigna la próxima referencia sin usar, y se aumenta el valor de ultima_ref en uno. Una vez que se tenga la referencia de la persona, se guarda la imagen en el disco duro con el formato de nombre de archivo ya comentado anteriormente, y se guarda en el archivo *imgref.txt* dicho nombre junto con la referencia y el nombre completo, y se marca al usuario como activo. Por último, se vuelve a llamar a la función aplicar_pca, ya que cada vez que se añade un nuevo usuario hay que volver a calcular los *eigenvectors* al haber sido modificado el espacio que forman todas las imágenes faciales.
- buscar_persona: esta función recibe como entrada el nombre completo de una persona almacenada en la base de datos, la compara con los nombres del vector *v_nombres* y, finalmente, si encuentra el nombre, devuelve su referencia asociada.
- agregar_foto: para facilitar el reconocimiento de algún usuario concreto, se pueden añadir más imágenes de dicha persona. Al llamar a esta función, se le pasa como entrada tanto la foto a añadir ya procesada como la referencia única del usuario. La función se encarga de

guardar la imagen en el disco duro y vuelve a aplicar el PCA con la nueva imagen añadida al espacio de imágenes faciales.

- borrar_persona: esta función recibe como parámetro de entrada la referencia de la persona que ya no será tenida en cuenta en el reconocimiento. Realmente no se borran ni las imágenes del disco duro ni las líneas correspondientes a ese imágenes en el archivo imgref.txt. Lo que se hace es cambiar el estado de dichas personas de A (activo) a B (borrado) en el archivo imgref.txt. Para ello, se abre dicho archivo en modo lectura con *ifstream*, y a la vez se abre en modo escritura un archivo temporal. A la vez que se recorren las líneas del primero, se van copiando en el segundo. De esta forma, cuando lleguemos a las líneas de la persona a borrar, se copiarán con el estado B. Por último, se borra el archivo original y se renombra el archivo temporal como *imgref.txt*. Una vez que se han realizado todos los pasos, se vuelven a calcular los *eigenvectors*.
- buscar_cara: la función buscar_cara es la parte principal del reconocimiento. Recibe como entrada una cara desconocida y devuelve la referencia de la persona de la base de datos reconocida. Además, es una función sobrecargada, por lo que también se le puede pasar un *string* por referencia, almacenando en este el nombre completo de dicha persona.

  Lo primero que se hace es transformar la imagen desconocida a una matriz de una sola columna. A esta matriz se le aplican los *eigenvectors* calculados previamente con las imágenes de entrenamiento. A continuación, se compara la imagen proyectada con las imágenes proyectadas de entrenamiento por medio de la distancia euclídea. Para hacer esto se recorre el vector *v_caras_pro* y se calcula la distancia euclídea de cada una de las matrices respecto de la imagen proyectada. De esta manera, cada vez que se calcula una distancia se compara para saber si es la más baja hasta ahora, y de ser así, se almacena el valor y la imagen a la que pertenece. Una vez recorrido todo el vector, se compara la menor distancia encontrada con un valor umbral que discierne entre los positivos y los falsos positivos. Para definir el umbral, se testea la aplicación con varias imágenes distintas, y se asigna el umbral que mejor relación positivo / falso positivo tiene. Un valor alto de umbral descartará muchos reconocimientos correctos, mientras que uno bajo dará lugar a muchos falsos positivos, incluso cuando la persona a reconocer y la persona de entrenamiento se parezcan poco. El código de la búsqueda de la menor distancia euclídea es el siguiente:

```
for (int iCara = 0; iCara < v_caras_pro.size(); iCara++)
{
	double dist_cuadrada = 0;
	for (int i = 0; i < total_eigenvectors; i++)
	{
		float d_i = cara_proyectada.col(0).at<float>(i) -
		v_caras_pro[iCara].col(0).at<float>(i);

// Distancia euclídea al cuadrado. No se calcula la raíz porque no es relevante

		dist_cuadrada += d_i*d_i;
	}
	if (dist_cuadrada < distancia_minima)
	{
		distancia_minima = dist_cuadrada;
		cara_parecida = iCara;
	}}
```

- total_imagenes_usuario: función que devuelve la cantidad de imágenes que tiene almacenadas un usuario concreto.
- test_reconocimiento: para comprobar la tasa de acierto de la aplicación y ajustar el valor umbral, se define la función test_reconocimiento. La función cargará en el objeto OBDCaras una base de datos de imágenes faciales de libre utilización. A continuación, aplicará el reconocimiento a un listado de imágenes faciales distintas a las del entrenamiento pero de las mismas personas. Como la función conoce la identidad de la persona, también conocerá si el reconocimiento ha sido positivo o en su lugar ha sido un falso positivo. Para hacer todo esto, la función se apoyará en dos ficheros similares al de *imgref.*txt que previamente habremos preparado para realizar el test. Uno de ellos contendrá las líneas correspondientes a las imágenes de entrenamiento y el otro a las imágenes de reconocimiento.

  Dentro de la carpeta imagenestest, hay varias carpetas con los nombres de las bases de datos que utilizaremos. Dentro de ellas están los archivos txt y las imágenes de la base de datos. Según la que queramos utilizar, tendremos que modificar la constante NOMBRE_BD_IMAGENES definida en el propio código del programa. También tendremos que modificar la constante ANCHO_CARA_TEST ya que los valores de píxeles del ancho de las imágenes varían según la base de datos usada.

  Para crear los txt que leen las imágenes de entrenamiento y de test, se han rellenado los datos previamente en una hoja de Excel. De esta forma, podemos aprovechar el rellenado automático para preparar rápidamente la hoja (figura 25). A continuación, esta hoja se guarda como archivo csv (*Comma Separated Values* o valores separados por comas) y al abrirla con el bloc de notas, podemos reemplazar la coma rápidamente por espacios o simplemente quitarla (figura 26). En la figura 27 podemos ver el fichero final que se utilizará en el test.

	A	B	C	D
1	1	testimages\Essex\	1	_2.pgm
2	1	testimages\Essex\	1	_3.pgm
3	2	testimages\Essex\	2	_2.pgm
4	2	testimages\Essex\	2	_3.pgm
5	3	testimages\Essex\	3	_2.pgm
6	3	testimages\Essex\	3	_3.pgm
7	4	testimages\Essex\	4	_2.pgm
8	4	testimages\Essex\	4	_3.pgm
9	5	testimages\Essex\	5	_2.pgm
10	5	testimages\Essex\	5	_3.pgm
11	6	testimages\Essex\	6	_2.pgm
12	6	testimages\Essex\	6	_3.pgm
13	7	testimages\Essex\	7	_2.pgm
14	7	testimages\Essex\	7	_3.pgm
15	8	testimages\Essex\	8	_2.pgm

Figura 25 Se rellena el contenido del txt en una hoja de cálculo

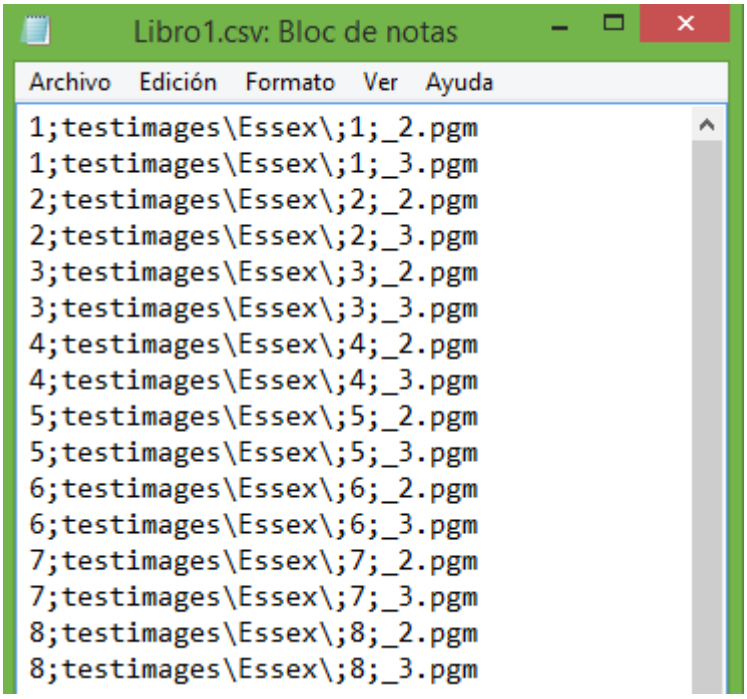


Figura 26 Apariencia del archivo csv abierto con bloc de notas. Basta sustituir los ; por espacios o eliminarlos

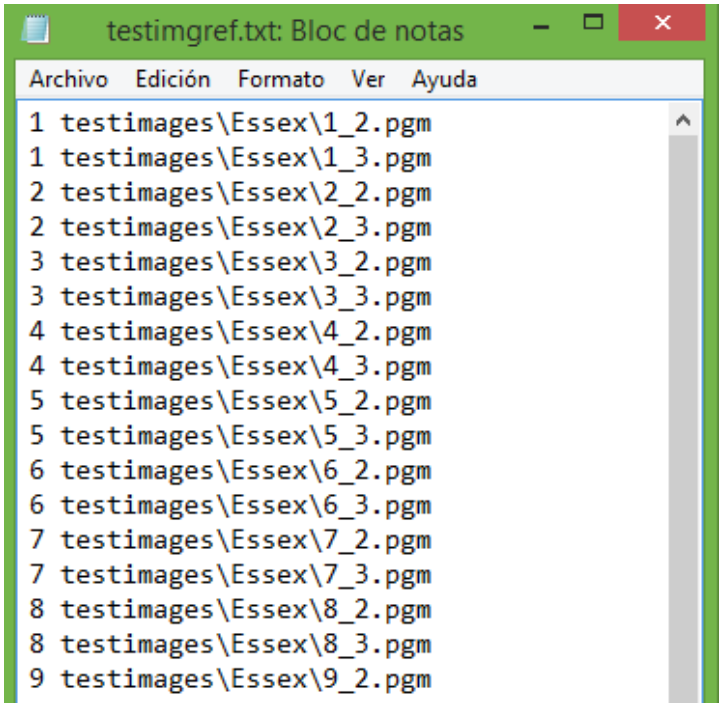


Figura 27 Apariencia final de archivo de test

Como se puede observar, las imágenes empleadas en los test están almacenadas en una ruta distinta de las imágenes que almacena la aplicación durante su funcionamiento normal.

Una vez finalizado el test, la aplicación muestra los resultados en pantalla (figura 28) y a continuación vuelve a cargar la base de datos normal de la aplicación.

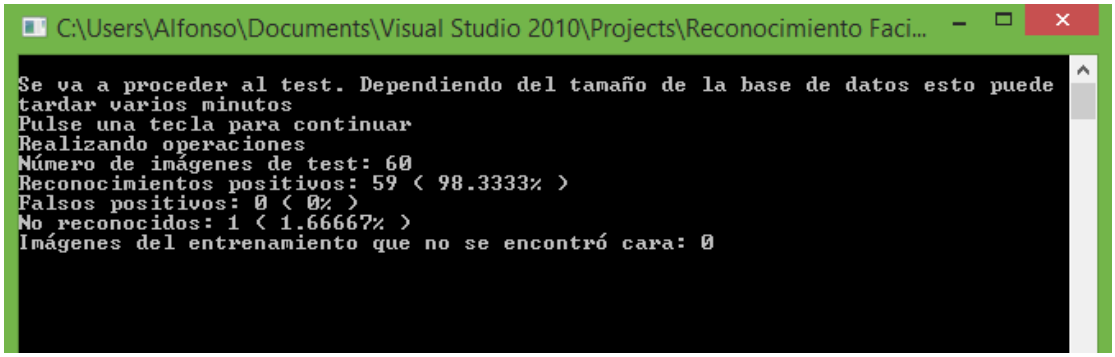


Figura 28 Resultados del test de reconocimiento

- mostrar_eigenfaces: Esta función no es necesaria para realizar el reconocimiento, pero se ha implementado para poder visualizar las *eigenfaces* según el estado actual de la base de datos. Los *eigenvectors* están disponibles en la clase pca que implementa OpenCV. Se puede acceder a ellos directamente. Para representarlos, primero hay que transformalos a las dimensiones de una imagen, ya que están almacenados como columnas, y a continuación normalizar sus valores entre 0 y 255 (para convertirlos en valores de gris). La función ha sido implementada para representar las 4 primeras *eigenfaces*, componiendo con ellas una sola imagen. En la figura 29 podemos ver una representación de las cuatro primeras *eigenfaces* a partir de 6 imágenes tomadas y procesadas por la aplicación. El código de transformación de los *eigenvectors* en imágenes se puede implementar en una sola línea:

```
normalize(pca.eigenvectors.row(0).reshape(1, v_caras[0].rows), eigen1, 0, 255,
NORM_MINMAX, CV_8UC1);
```

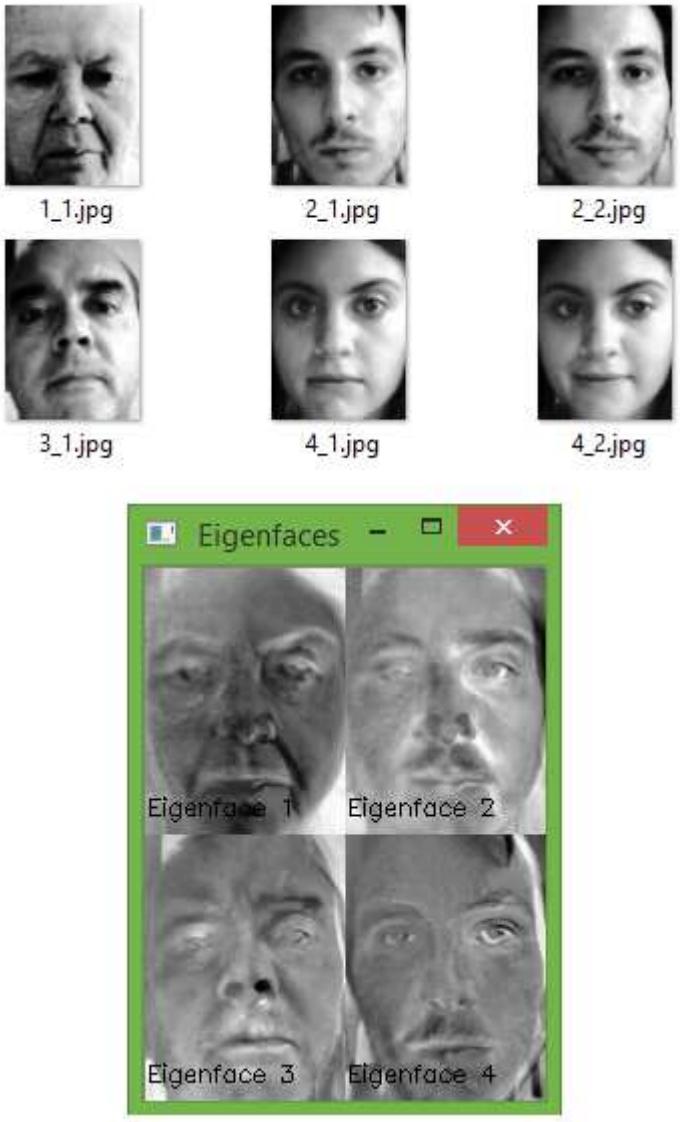


Figura 29 Ejemplo de imágenes faciales y sus *eigenfaces* representadas como imágenes

6.6 Integración y uso

Una vez vistos por separado las distintas partes de la aplicación, queda integrarlas en la aplicación principal. Para ello se define el fichero Reconocimiento Facial.cpp. Esté será el punto de entrada a la aplicación.

Al iniciar la aplicación, lo primero que se hace es crear el objeto OBDCaras y cargarle los datos leyendo el archivo imgref.txt. De esta forma, ya tendremos todas las imágenes de entrenamiento proyectadas a los *eigenvectors*. Solo será necesario aplicar de nuevo el PCA en caso de que se añadan, modifiquen o borren nuevas imágenes faciales.

En caso de que la aplicación no encuentra el fichero imgref.txt (bien porque se haya borrado o porque sea la primera vez que se ejecuta), automáticamente crea el fichero vacío y crea la carpeta donde se almacenarán las imágenes.

Una vez se han cargado las imágenes en la memoria y se ha aplicado el PCA, en la consola de comandos aparece un menú en el que podremos elegir la acción realizar, todas ellas directamente relacionadas con las funciones de la clase BDCaras (ver figura 30). Hay que recordar que siempre que se añada, borre o modifique una imagen, la propia función del objeto OBDCaras inicializará de nuevo sus atributos y aplicará de nuevo el PCA teniendo en cuenta el nuevo estado de la base de datos.

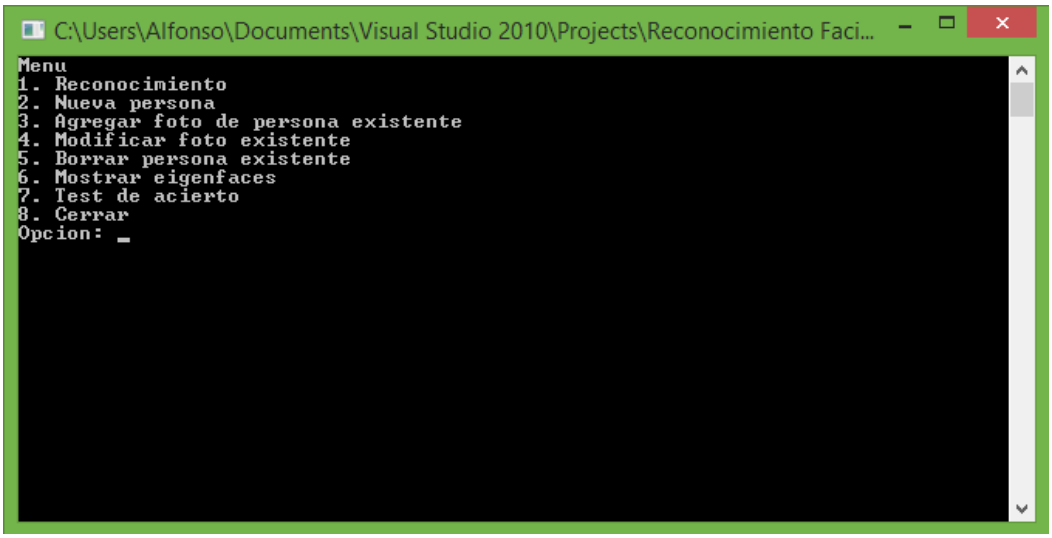


Figura 30 Ejemplo de inicio de programa

1- Reconocimiento: La pantalla abre una ventana de Windows en la que se reproducen continuamente las imágenes tomadas desde la cámara configurada por defecto mientras que busca caras en cada uno de los *frames*. En caso de que encuentre alguna, la rodeará con un círculo y aplicará la función de reconocimiento (figura 31). Si encuentra una persona en la base de datos que se parezca lo suficiente, cerrará la ventana y mostrará en la consola el nombre de la persona (figura 32). Si no encuentra ningún parecido, seguirá leyendo imágenes hasta que la encuentre o el usuario cancele pulsando ESC.

Figura 31 Se ha encontrado una cara. La aplicación está buscando una cara parecida

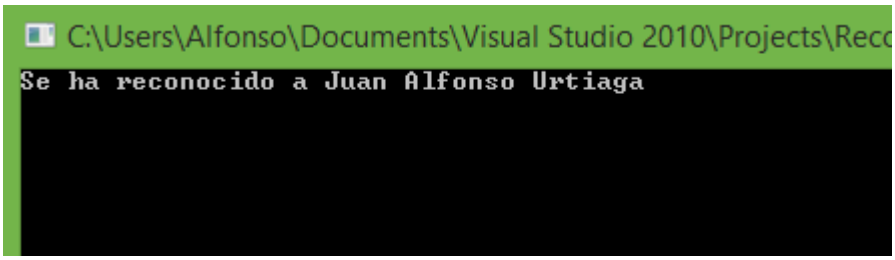


Figura 32 La aplicación muestra el nombre de la persona reconocida

2- Nueva persona: Con esta opción podremos añadir nuevos sujetos a la base de datos. Primero se nos pide que se introduzca el nombre de la persona (figura 33). El nombre no debe llevar acentos ni la letra "ñ" por las limitaciones de c++ y el formato de caracteres que emplea. Una vez introducido, se abre una ventana similar a la del reconocimiento, solo que en esta ocasión el programa espera a que el usuario se posicione correctamente y pulse Intro para mantener una imagen congelada en pantalla (figura 34). Hay que pulsar Intro una segunda vez para confirmar que la imagen congelada es buena para el entrenamiento (figura 35).

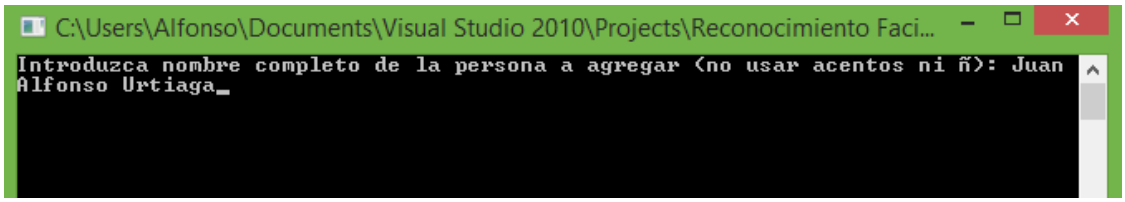


Figura 33 Introducción de nombre

Figura 34 La aplicación espera a que se posicione el usuario y confirme con Intro

Figura 35 Imagen congelada a la espera de confirmación

3- Agregar foto de persona existente: Igual que al agregar una nueva persona, la aplicación pide al usuario que introduzca el nombre de la persona, solo que en esta ocasión llamará a la función buscar_persona del objeto OBDCaras y buscará la referencia que concuerde con el nombre introducido. Si no encuentra nadie con el mismo nombre, mostrará un mensaje de error y volverá al menú. En caso de encontrarlo, el procedimiento de la captura de la imagen es el mismo que en la opción agregar persona.
4- Modificar foto existente: la aplicación pide al usuario el nombre de la persona. Si encuentra dicho nombre, muestra cuántas imágenes tiene almacenadas dicho usuario. A continuación, se debe introducir el número de la imagen a modificar (figura 36). Después, abre una ventana de Windows con las imágenes de la cámara de idéntica manera a cómo se ha hecho antes. Por último, se sobrescribe la imagen elegida con la tomada por la cámara.

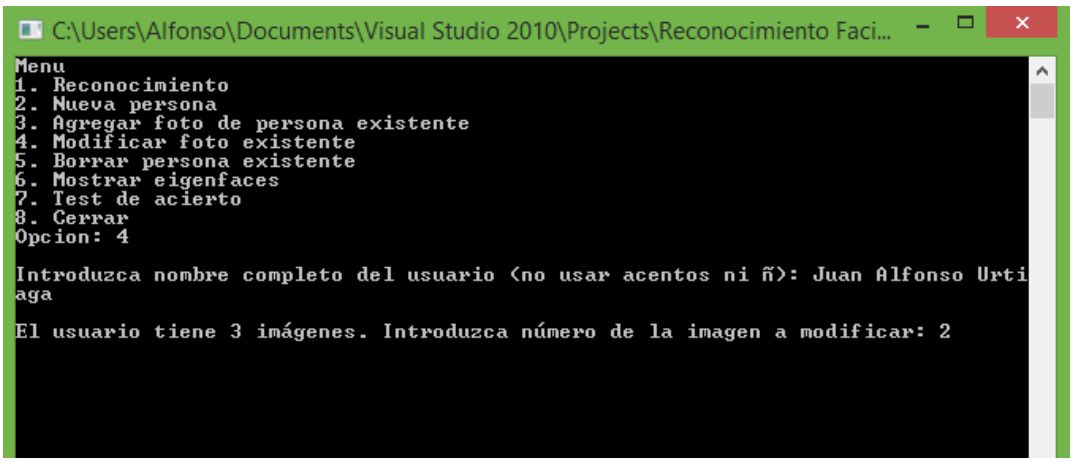


Figura 36 Modificación de foto existente

5- Borrar persona existente: Se ha de introducir el nombre de la persona a borrar. Si el nombre está en la base de datos, se modificará el archivo imgref.txt para que todas las imágenes faciales asociadas a dicho usuario aparezcan marcadas con B (Borrado). Sin embargo, las imágenes no son borradas realmente de la carpeta, aunque no serán tenidas al aplicar el PCA ni el reconocimiento.
6- Mostrar eigenfaces: la aplicación llama a la función mostrar_eigenfaces del objeto OBDCaras, que muestra en una ventana las 4 primeras *eigenfaces* según la situación actual de la base de datos.
7- Test de acierto: la aplicación llama a la función test_reconocimiento del objeto OBDCaras. Como se comentaba anteriormente, esta opción está enfocada a ser utilizada cuando se depure la aplicación desde el IDE, ya que los parámetros del test (base de datos a utilizar y ancho de píxeles de las caras a buscar) están implementadas en el propio código de la aplicación. En caso de generar la aplicación para el usuario final, se debería quitar esta opción del menú. La opción está en el menú porque durante el desarrollo es útil poder ejecutarla durante las pruebas de ajuste de umbral.

7 RESULTADOS

7.1 Test de reconocimiento

En el apartado de implementación se habla de un valor umbral que no debe ser superado para dar como bueno el reconocimiento. Para definir dicho valor umbral, usaremos la función de test implementada en la aplicación. Se realizará varias veces con distintas bases de datos y con distintos valores de umbral. De esta forma, podremos comparar los resultados y elegir un valor umbral que tenga una tasa de falsos positivos baja sin sacrificar por ello mucho porcentaje de acierto.

Para testear la aplicación se utilizan tres bases de datos distintas: la de la Universidad de Essex, la de AT&T Laboratories y la de BioID Web Services.

7.1.1 Universidad de Essex

La base de datos de la Universidad de Essex, publicado por el Dr. Libor Spacek en su página web [18], es una base de datos compuesta por cuatro conjuntos de imágenes de dificultad creciente. Para el test utilizaremos imágenes de los dos primeros conjuntos. Estos están formados por imágenes capturadas en condiciones de iluminación controladas, con muy pequeñas variaciones de pose (figura 37).

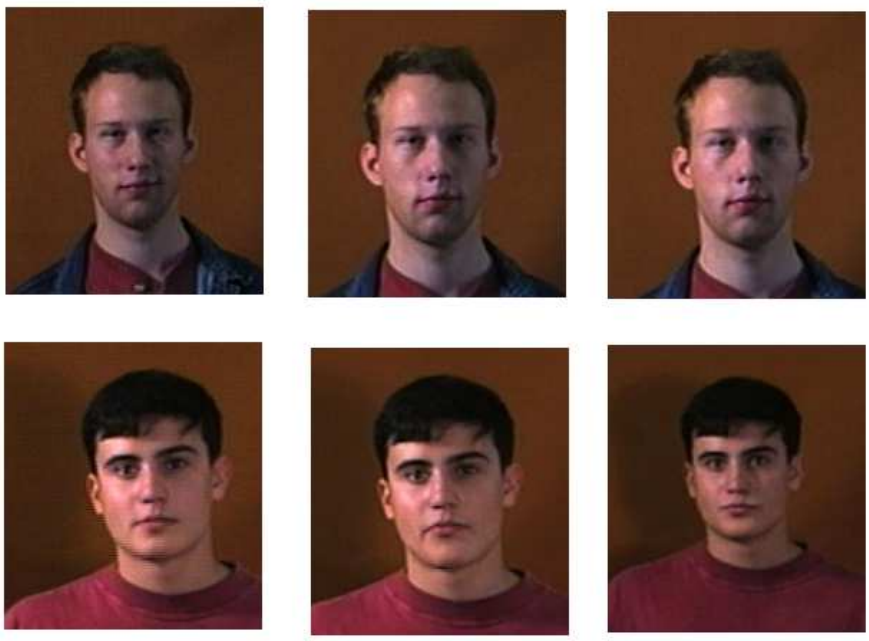


Figura 37 Ejemplos de imágenes de la Universidad de Essex

7.1.2 AT&T Laboratories

La base de datos de AT&T [19] está formada por imágenes faciales de 92x112 píxeles, de 40 personas diferentes, tomadas entre Abril de 1992 y Abril de 1994. De cada sujeto se tomaron 10 imágenes distintas, variando entre ellas la iluminación, la orientación, la expresión, la fecha de la toma e incluso algunas con gafas. Las imágenes fueron tomadas frente a un fondo negro homogéneo (figura 38).

Figura 38 Ejemplo de imágenes de AT&T

7.1.3 BioID Web Services

La base de datos de BioID [20] contiene 1521 imágenes faciales de 23 personas diferentes con una resolución de 384x286 píxeles. El enfoque principal de la base de datos es el de recrear situaciones de uso reales, por lo tanto hay variaciones en la iluminación, pose, fondo y tamaño de la cara. Para el test de nuestra aplicación se escogen cuatro imágenes de 22 sujetos, tres de ellas para la fase de entrenamiento y la cuarta será una imagen tomada un día distinto para el reconocimiento. Las tres imágenes de entrenamiento tendrán variaciones entre sí (figura 39).

Figura 39 Ejemplo de imágenes de BioID

7.2 Resultados de los test

Lo primero a tener en cuenta a la hora de realizar los test es la variedad de tamaños de las imágenes en píxeles. Para este fin se han definido dos constantes en el archivo funcreconocimiento.cpp, ANCHO_CARA_TEST y ALTO_CARA_TEST. Igual que en la implementación del reconocimiento, el valor de ALTO_CARA_TEST es proporcional al de ANCHO_CARA_TEST, por lo que sólo hay que modificar uno de los valores. Para las pruebas, debido al tamaño de las imágenes disponibles, se define el ancho de la cara en 50 píxeles, siendo de 66 píxeles el alto

Los resultados de los diferentes test son los siguientes. Positivo es que el reconocimiento es correcto, falso positivo es que se ha reconocido un sujeto distinto, negativo es que no se ha encontrado ninguna cara que supere el umbral y el total de imágenes es la cantidad de imágenes empleadas en el reconocimiento.

- Para los primeros test se usa la base de datos de la Universidad de Essex. Esta base de datos tiene una dificultad baja, ya que hay poca variación entre las imágenes del mismo sujeto. Para la realización del test utilizaremos imágenes de 30 sujetos distintos. De cada sujeto emplearemos 3 imágenes, una para el entrenamiento y dos para reconocimiento. Los resultados figuran en la tabla 2.

Tabla 2 Resultados con la base de datos de Essex

Umbral	Positivo	Falso positivo	Negativo	Total imágenes
50	95%	0%	5%	60
80	98,3%	0%	1,7%	60
100	98,3%	0%	1,7%	60
120	100%	0%	0%	60

Como se puede observar por los resultados, para situaciones de iluminación controladas y poca variación de pose, la técnica de reconocimiento de PCA es efectiva 100%.

- En el test con la base de datos de AT&T, a la hora valorar los resultados hay que tener en cuenta que en esta base de datos hay mucha variación entre imágenes del mismo sujeto debido a cambios de pose, luz y expresión. Además, si en la fase de entrenamiento no se encuentra ninguna cara, no se almacenara dicho sujeto, lo que disminuye la tasa de acierto. Para corregir un poco esta tendencia, de las 10 imágenes de cada sujeto se utilizan las dos primeras en el entrenamiento y el resto para el reconocimiento. Durante la fase de entrenamiento, el algoritmo no encontró caras en 19 imágenes de las 80 suministradas. En la tabla 3 aparecen los resultados para esta base de datos

Tabla 3 Resultados con la base de datos de AT&T

Umbral	Positivo	Falso positivo	Negativo	Total imágenes
50	33,2%	3,2%	63,6%	253
80	49,8%	16,2%	34%	253
100	54,9%	23,7%	21,4%	253
120	58,5%	29,6%	11,9%	253
140	61,3%	33,6%	5,1%	253
160	62%	35,1%	2,7%	253
200	62%	37,9%	0%	253
250	62%	37,9%	0%	253

En los resultados podemos apreciar que para valores de umbral superiores a 140 apenas hay cambios en los porcentajes de acierto. En el lado opuesto, para valores inferiores a 80 la tasa de reconocimientos negativos (no se encuentra ninguna imagen facial parecida) es demasiada alta, superior al 50%, de lo que se deduce que el valor de umbral se debe ajustar entre 80 y 140.

- Para realizar el test con la base de datos de BioID, se han tomado tres fotos de cada usuario, dos se emplean en el entrenamiento y la tercera como reconocimiento. En la tabla 4 podemos apreciar que los resultados del PCA son muy pobres cuando se presentan demasiadas variaciones en las condiciones de toma de imagen. También hay que tener en cuenta que los usuarios de esta base de datos no están buscando facilitar el reconocimiento ya que las imágenes se han tomado como si los sujetos estuvieran manteniendo conversaciones por la webcam en días distintos.

Tabla 4 Resultados con la base de datos de BioID

Umbral	Positivo	Falso positivo	Negativo	Total imágenes
80	31,8%	0%	68,2%	22
100	36,4%	9%	54,5%	22
120	36,4%	13,6%	50%	22
140	40,9%	27,3%	31,8%	22
160	40,9%	50%	9%	22
200	45,4%	54,5%	0%	22

7.3 Valor umbral

Para elegir un valor umbral adecuado, hay que fijarse en el cambio que acarrea en los tres valores (positivos, falsos positivos y negativos). También hay que tener en cuenta que la aplicación que se ha desarrollado está orientada a una situación de iluminación controlada con la colaboración del sujeto a identificar, ya que para ello se ha implementado la necesidad de confirmar la imagen como válida durante el entrenamiento. Además, el reconocimiento se realiza de manera continuada leyendo los *frames* de entrada de la webcam. Esto hace que sea aceptable una tasa de negativos alta, ya que el sujeto puede colocarse frente a la cámara intentando facilitar el reconocimiento, mientras que los falsos positivos son imposibles de evitar una vez que se han realizado.

Por todo ello, se ha elegido como valor umbral 100, que a pesar de tener valores negativos altos, mantiene una tasa baja de falsos positivos.

8 CONCLUSIONES

En el presente Proyecto Fin de Carrera se han visto varias técnicas de reconocimiento facial y se ha profundizado en una de ellas, el PCA.

A la vista de los resultados obtenidos y de los artículos estudiados, podemos concluir con que el PCA por sí solo no es una técnica sólida de reconocimiento facial: depende demasiado de las condiciones ambientales, de la posición y orientación del sujeto, y de la expresión facial que este adopte. Al ser una técnica basada exclusivamente en los valores de gris de una imagen, el reconocimiento no se basa tanto en los detalles fisionómicos de la cara sino en la forma que la luz incide sobre ella. Dos imágenes faciales del mismo sujeto pueden ser completamente distintas para el PCA a pesar de estar tomadas con la misma orientación y expresión facial si la iluminación no es la misma en ambas, mientras que dos imágenes de dos personas distintas con una iluminación similar en la imagen pueden ser detectadas como pertenecientes a la misma persona. Esto hace que el PCA sólo sea útil en ambientes con iluminación controlada, ya que es muy sensible a pequeños cambios respecto de las imágenes de entrenamiento.

Durante las pruebas realizadas con la aplicación, se comprobó como una simple ventana puede dificultar enormemente el reconocimiento. A lo largo del día, la iluminación de una habitación con ventana cambia continuamente, y para el PCA las imágenes tomadas durante la mañana pueden ser completamente diferentes a las del mediodía o a las de la tarde.

Para poder utilizar la aplicación desarrollada en el presente proyecto con una buena relación de positivos / falsos positivos, la iluminación debe ser frontal, directa y controlada, de forma que elementos externos de iluminación no produzcan sombras en las caras. La orientación de la cara y la expresión facial también dificultan el reconocimiento, pero a un nivel mucho menor que la iluminación. Además, estas dificultades pueden ser salvadas con la colaboración de los usuarios poniendo una expresión neutra tanto en el entrenamiento como en el reconocimiento.

El fondo de las imágenes no afecta al reconocimiento, ya que la aplicación recorta las caras de las imágenes y el fondo no es procesado como información. En todo caso, podría afectar a la detección facial, pero las pruebas realizadas indican que si el tamaño de la cara en píxeles es suficiente y la orientación de la cara entra dentro de unos márgenes aceptables, la aplicación detectará la cara de la persona.

A pesar de que por sí solo el PCA no es una técnica robusta de reconocimiento, es importante conocerla ya que técnicas más avanzadas la utilizan como herramienta estadística en la reducción de datos y análisis de patrones. El LDA no deja de ser una versión más compleja del PCA, en la que no sólo se miran imágenes independientes en el entrenamiento, sino que del conjunto de varias imágenes de un mismo sujeto se pueden extraer características comunes a todas ellas que facilitaran el reconocimiento del individuo.

En algunas de las pruebas realizadas con la aplicación desarrollada se han empleado varias imágenes del mismo sujeto en la fase de entrenamiento. El problema es que estas imágenes no están relacionadas entre sí, como sí pasa con el LDA, sino que se intenta buscar varios escenarios distintos que se pueden dar a la hora del reconocimiento para que sea más fácil encontrar una imagen semejante a la tomada.

Como posibles mejoras futuras de la aplicación, las principales pasarían por utilizar una técnica de reconocimiento alternativa como puede ser el LDA. También se puede mejorar la gestión de la base de datos. Como se comentó de pasada en el capítulo de Desarrollo, a la hora de diseñar la aplicación se valoró el que tuviera una base de datos SQL asociada. Finalmente, tras muchas pruebas y muchos errores de compilación desconocidos, se descartó esta posibilidad por no tener los conocimientos necesarios para gestionar una base de datos de SQL desde una aplicación de C++. Aun así, se mantuvo un formato similar al de una base de datos para la creación y uso del fichero imgref.txt, por lo que otra persona preparada para ello podría sustituir en poco tiempo las partes de la aplicación que se necesiten, añadiendo campos útiles para la gestión como las fechas de creación de las imágenes de entrenamiento o la fecha de la última vez que se detectó a un determinado usuario. El último punto que se podría mejorar es añadir una interfaz de usuario en lugar de utilizar la consola de comandos.

9 REFERENCIAS

[1] Paul Viola y Michael Jones, *Rapid Object Detection using a Boosted Cascade of Simple Features*, 2001

[2] Kresimir Delac and Mislav Grgic, *Face Recognition*, pp 94-97, 2007

[3] Kresimir Delac and Mislav Grgic, *Face Recognition*, pp 98-101, 2007

[4] Kresimir Delac and Mislav Grgic, *Face Recognition*, pp 101, 2007

[5] Chao Li y Armando Barreto, *An integrated 3D face-expression recognition approach*, 2006

[6] Sen Wang, Lei Zhang, and Dimitris Samaras, *Face Reconstruction Across Different Poses and Arbitrary Illumination Conditions*, 2005

[7] M. Tistarelli y E. Grosso, *Active vision-based face authentication*, 2000

[8] V. E. Neagoe, A. D. Ropot y A. C. Mugioiu, *Real Time Face Recognition Using Decision Fusion of Neural Classifiers in the Visible and Thermal Infrared Spectrum*, 2007

[9] Mrinal Kanti Bhowmik, Debotosh Bhattacharjee, Mita Nasipuri, Dipak Kumar Basu y Mahantapas Kundu, *Classification of polar-thermal eigenfaces using multilayer perceptron for human face recognition*, 2008

[10] Siu-Yeung Cho, Lingyu Wang y Wen Jin Ong, *Thermal Imprint Feature Analysis for Face Recognition*, 2009

[11] Wikipedia, *OpenCV* [http://es.wikipedia.org/wiki/OpenCV], consultado en Marzo 2014

[12] ISO/IEC 19794 *Information technology — Biometric data interchange formats — Part 5: Face image data*, ISO/IEC 19794-5, 2005

[13] OpenCV [http://opencv.org/], consultado en Marzo 2014

[14] Stack Overflow [http://stackoverflow.com], consultado Diciembre 2013

[15] Robin Hewitt, *Seeing With OpenCV, Part 4: Face Recognition With Eigenface*, [http://www.cognotics.com/opencv/servo_2007_series/part_4/index.html], consultado en Agosto 2013

[16] Robin Hewitt, *Seeing With OpenCV, Part 5: Implementing Eigenface*, [http://www.cognotics.com/opencv/servo_2007_series/part_5/index.html], consultado en Agosto 2013

[17] Shervin Emami [http://www.shervinemami.info/faceRecognition.html], consultado en Agosto 2013

[18] Dr Libor Spacek, [http://cswww.essex.ac.uk/mv/allfaces/index.html], consultado en Marzo 2014

[19] AT&T Laboratories [http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html], consultado en Marzo 2014

[20] BioID Web Services [http://www.bioid.com/index.php?q=downloads/software/bioid-face-database.html], consultado en Marzo 2014

Anexo A. Configuración OpenCV en Visual Studio 2010

La instalación y configuración de OpenCV en Visual Studio 2010 es relativamente sencilla gracias a los tutoriales disponibles en su web [http://docs.opencv.org/doc/tutorials/tutorials.html] (figura 40).

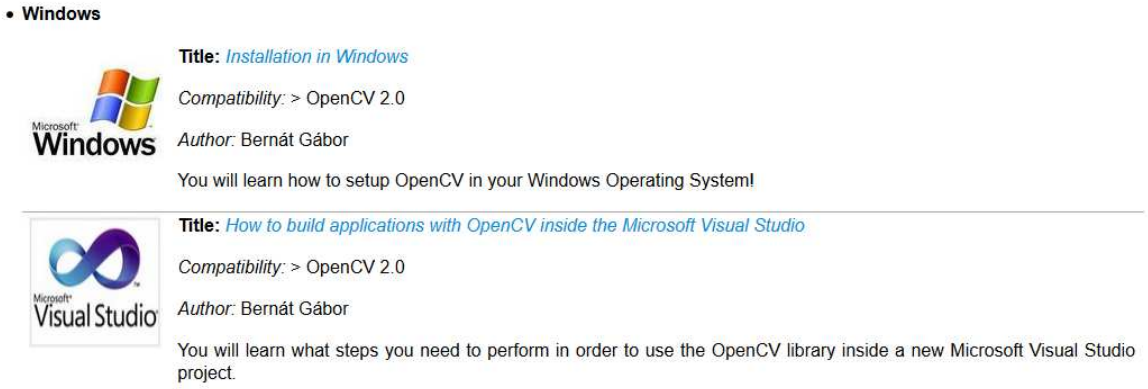


Figura 40 Tutoriales disponibles en la web opencv.org

El primero de ellos es referente a la instalación en Windows. Hay dos maneras de hacerlo, la primera es utilizando las librerías pre-compiladas, pero solo están disponibles si nuestro IDE es Visual Studio. La segunda es generando nuestras propias librerías desde los ficheros fuente. En nuestro caso utilizaremos directamente las librerías ya compiladas. En este apartado de la memoria, describiremos los pasos que se han seguido para la configuración, incluyendo algunos pasos adicionales que se tuvieron que dar y no aparecen reflejados en los tutoriales.

En la propia página de OpenCV, podremos acceder a las descargas de las distintas versiones de OpenCV disponibles en sourceorge.net [http://sourceforge.net/projects/opencvlibrary/files/opencv-win/] (figura 41). Actualmente está disponible hasta la versión 2.4.9. El presente proyecto empezó con la versión 2.4.2, pero fue recientemente actualizado a la 2.4.8.

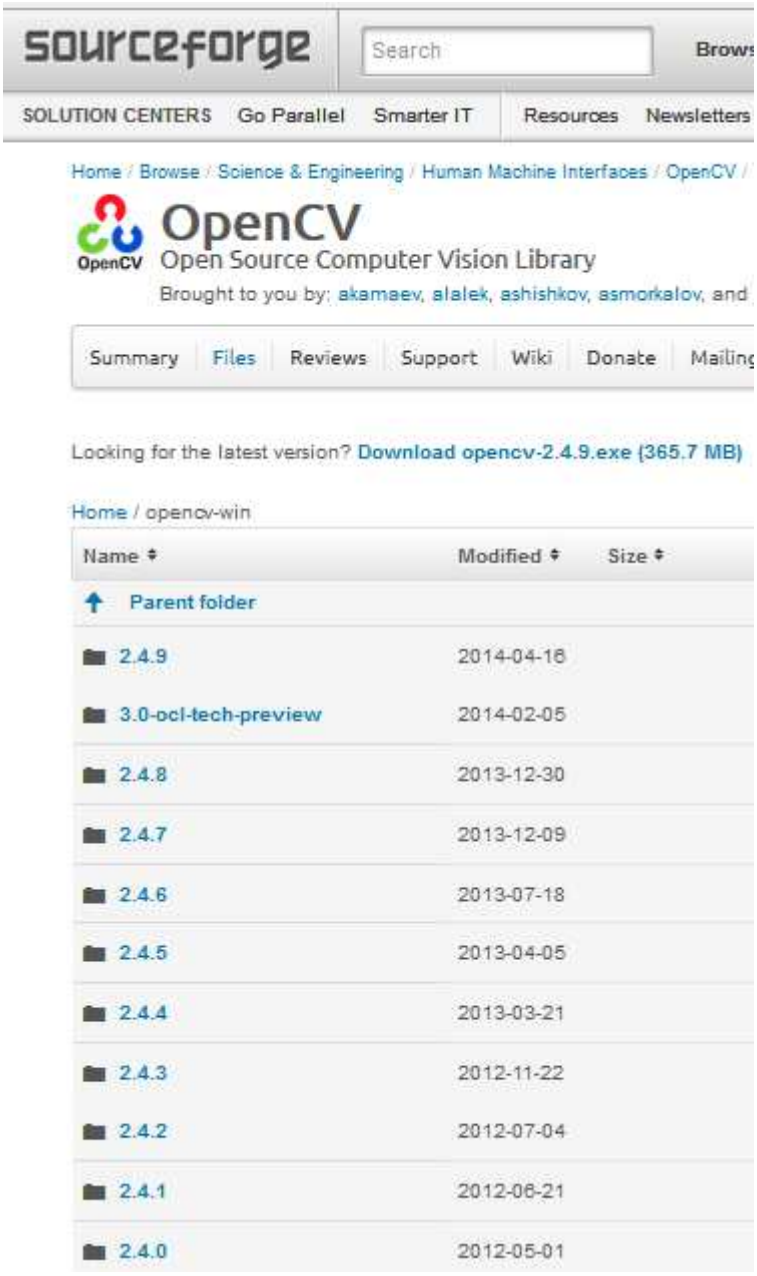


Figura 41 Distintas versiones de OpenCV disponibles en sourceforge.net

Una vez descargado el ejecutable, se instala en **c:/opencv248**. En esa ruta tendremos las librerías pre-compiladas para Visual Studio, en concreto para las versiones de 2010, 2012 y 2013. Dependiendo de qué versión utilicemos, y de si vamos generar una aplicación de 32 o 64 bits, utilizaremos unas u otras librerías. Como la versión empleada es la que tenemos disponibles para estudiantes en la web de Microsoft, se han utilizado las librerías de 86 bits para Visual 2010. Dentro de estas, hay dos tipos de librerías, las dinámicas y las estáticas.

La principal ventaja de las librerías estáticas frente a las dinámicas es que es más sencillo mover la aplicación a otros ordenadores ya que no es necesario instalar OpenCV en el ordenador que use la aplicación ni copiar las librerías. Su inconveniente es un mayor tamaño en el archivo ejecutable.

En el proyecto se ha optado por utilizar librerías estáticas ya que la idea es que la aplicación se pueda ejecutar en cualquier ordenador. La ruta completa de estas librerías es la siguiente:

C:\opencv248\build\x86\vc10\staticlib

Para facilitar la configuración en Visual Studio, se ha de añadir a las variables de entorno la variable **OPENCV_DIR** (figura 42), cuyo valor es:

C:\opencv248\build\x86\vc10

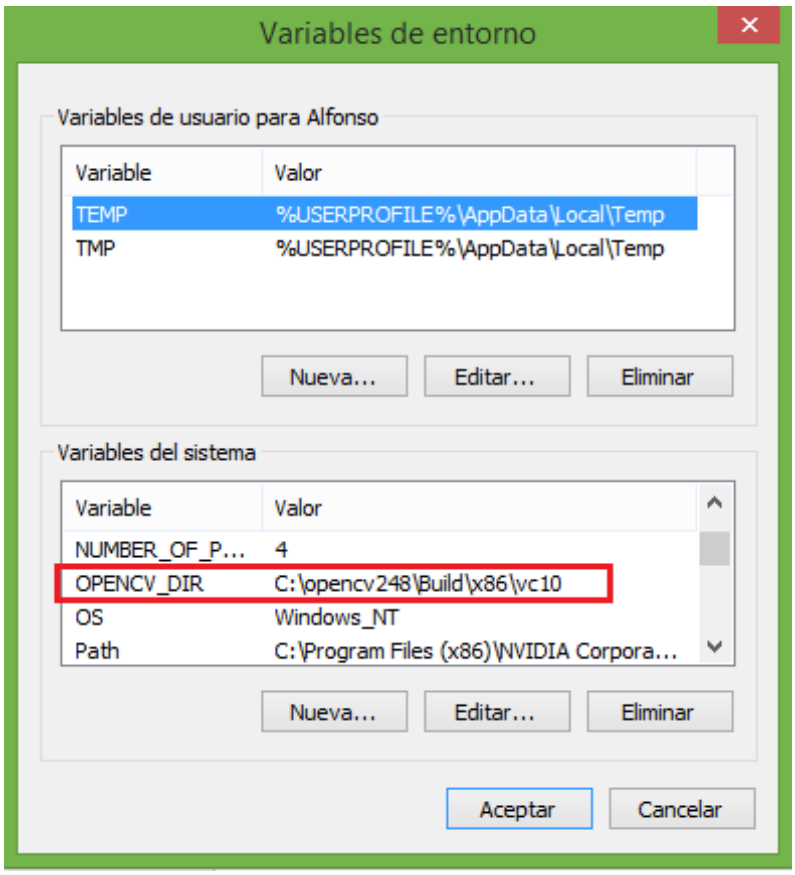


Figura 42 Variable de entorno OPENCV_DIR

El siguiente paso es configurar la solución en Visual Studio. En la página principal se crea un nuevo proyecto en blanco de tipo Aplicación de Consola Win32 (figuras 43 y 44).

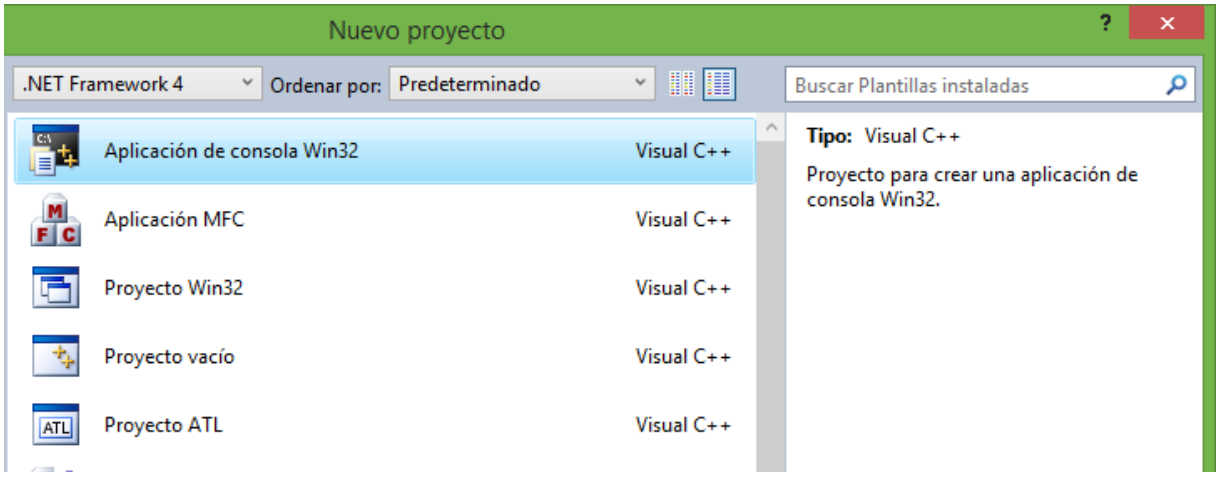


Figura 43 Menú nuevo proyecto

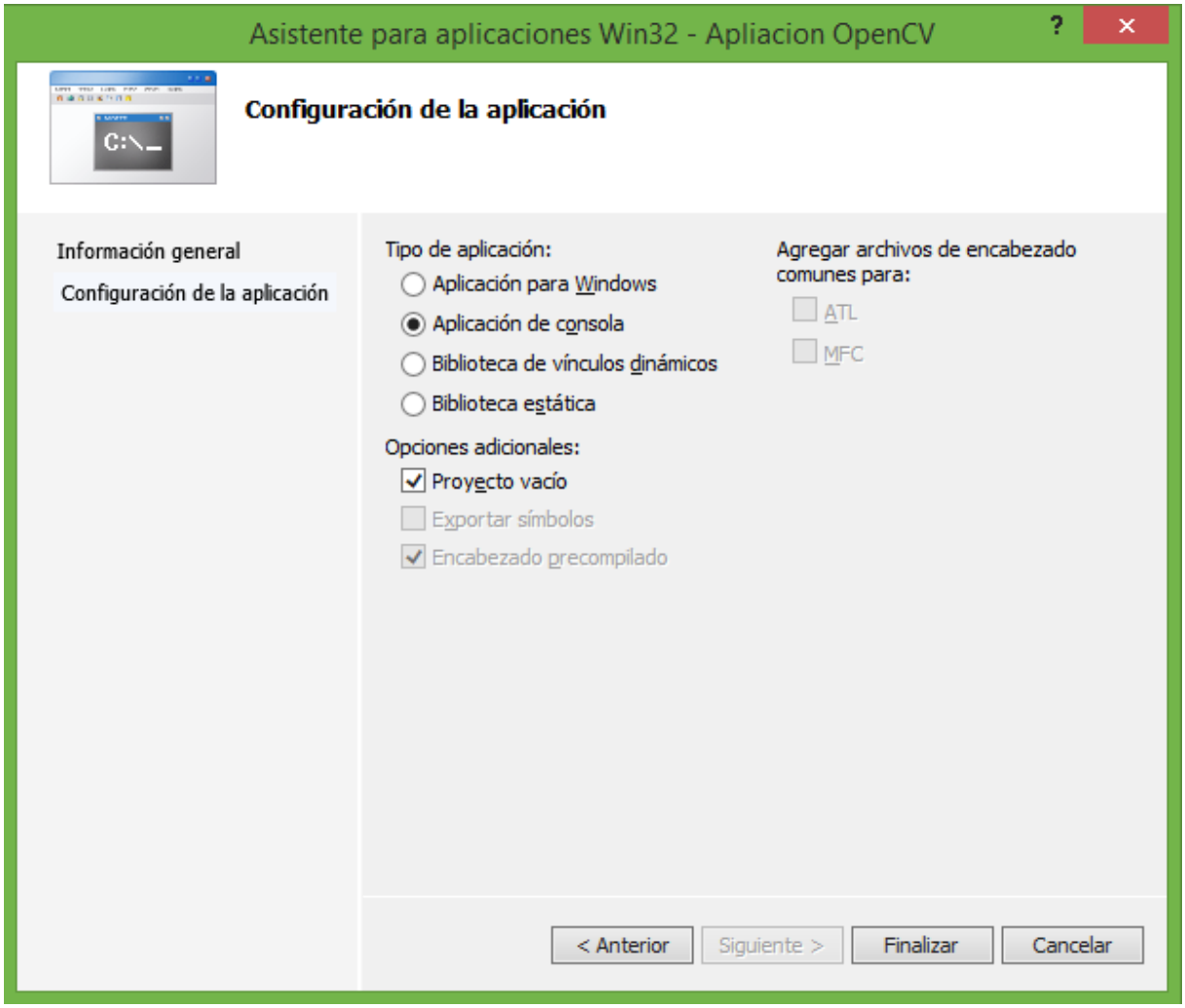


Figura 44 Configuración de la aplicación con Proyecto vacío marcado

En este nuevo proyecto, desde el Administrador de propiedades se crean dos nuevas hojas que llamaremos **OpenCV_Debug** y **OpenCV_Release** en sus carpetas respectivas (*Debug* y *Release,* ver figura 45).

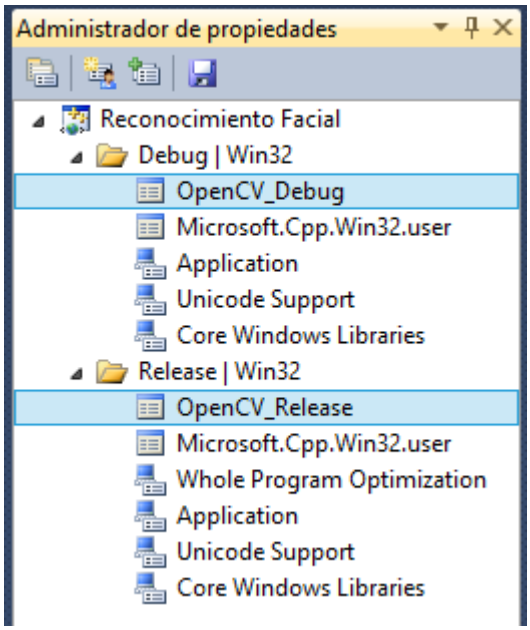


Figura 45 Hojas de propiedades *OpenCV_Debug* y *OpenCV_Release*

La configuración en ambas para aplicaciones con librerías estáticas es idéntica, diferenciándose solo en el último paso. Entrando en las hojas de propiedades, en *Propiedades comunes -> Directorios de VC++ -> Directorios de archivos de bibliotecas*, se añade **$(OPENCV_DIR)\staticlib** (figura 46). El valor **$(OPENCV_DIR)** será sustituido en tiempo de ejecución por la ruta que se ha definido en la variable de entorno, lo que hace más sencillo el portar la solución a otros ordenadores ya que no dependerá de la carpeta concreta en la que se hayan instalado las librerías.

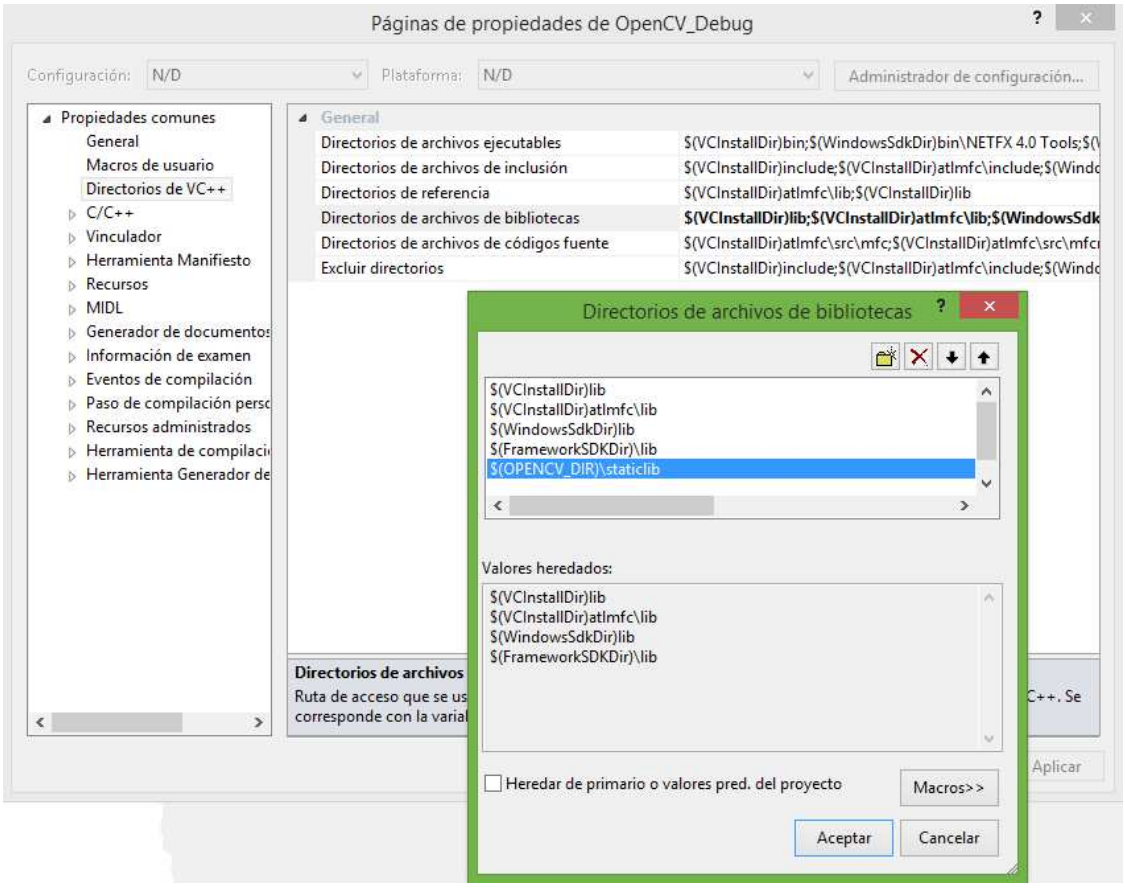


Figura 46 Directorio de bibliotecas

Siguiendo con la configuración de la hoja de propiedades **OpenCV_Debug**, en *Propiedades comunes -> C++ -> Directorios de inclusión adicionales* (Figura 47), se añade la línea **$(OPENCV_DIR)\..\..\include**

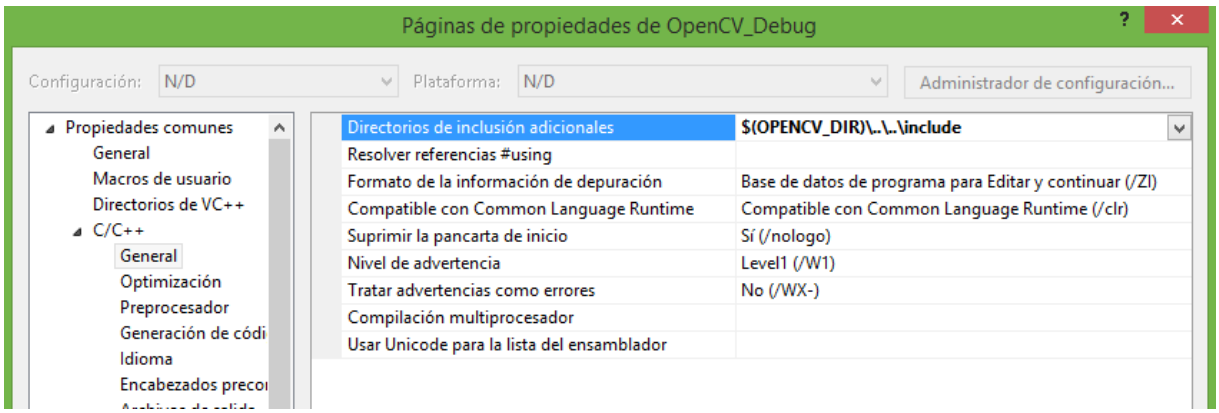


Figura 47 Directorios de inclusión adicionales

Por último, queda por añadir el listado de librerías. Esto se hace en *Propiedades comunes -> Vinculador -> Entrada -> Dependencias adicionales (*figura 48). Aquí se añaden las librerías de OpenCV que estén en la carpeta staticlib. De cada librería hay dos archivos diferenciados por una *d* al final del nombre. En la hoja **OpenCV_Debug** añadiremos los archivos con la letra *d*, mientras que en **OpenCV_Release** añadiremos las mismas librerías pero sin la *d*.

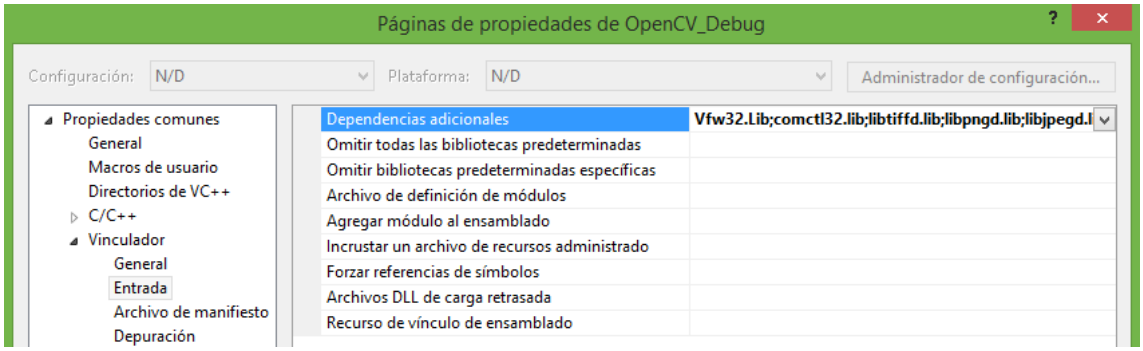


Figura 48 Dependencias adicionales

El listado de las librerías añadidas en la hoja **OpenCV_Debug** es el siguiente:

```
Vfw32.Lib
comctl32.lib
libtiffd.lib
libpngd.lib
libjpegd.lib
libjasperd.lib
IlmImfd.lib
zlibd.lib
opencv_calib3d248d.lib
opencv_contrib248d.lib
opencv_core248d.lib
opencv_features2d248d.lib
opencv_flann248d.lib
opencv_gpu248d.lib
opencv_highgui248d.lib
opencv_imgproc248d.lib
opencv_legacy248d.lib
opencv_ml248d.lib
opencv_nonfree248d.lib
opencv_objdetect248d.lib
opencv_ocl248d.lib
opencv_photo248d.lib
opencv_stitching248d.lib
opencv_superres248d.lib
opencv_ts248d.lib
opencv_video248d.lib
opencv_videostab248d.lib
```

Figura 49 Librerías añadidas en la hoja OpenCV_Debug

Para terminar con la configuración según los tutoriales oficiales habría que realizar los mismos pasos para la hoja **OpenCV_Release**, quitando la *d* final en las librerías que la lleven. En principio con esto bastaría para poder usar OpenCV, pero debido a algunos errores que aparecían a la hora de compilar, he tenido que hacer además las siguientes modificaciones en ambas hojas de propiedades:

En *Propiedades comunes -> C++ -> General -> Compatible con Common Language Runtime* (figura 50), elegir la opción *No es compatible con Common Language Runtime*.

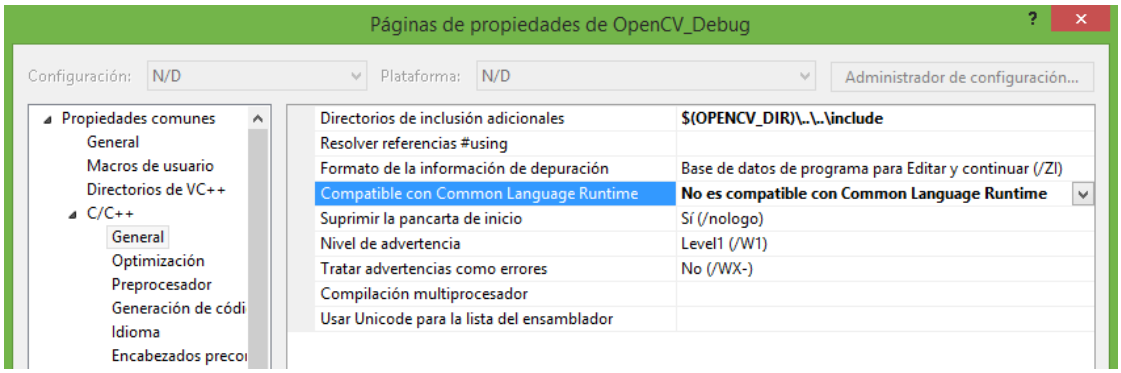


Figura 50 Elegir No es compatible con Common Language Runtime

En *Propiedades comunes -> C++ -> Optimización -> Biblioteca en tiempo de ejecución* (figura 51), elegir la opción **Depuración multiproceso (/MTd)**.

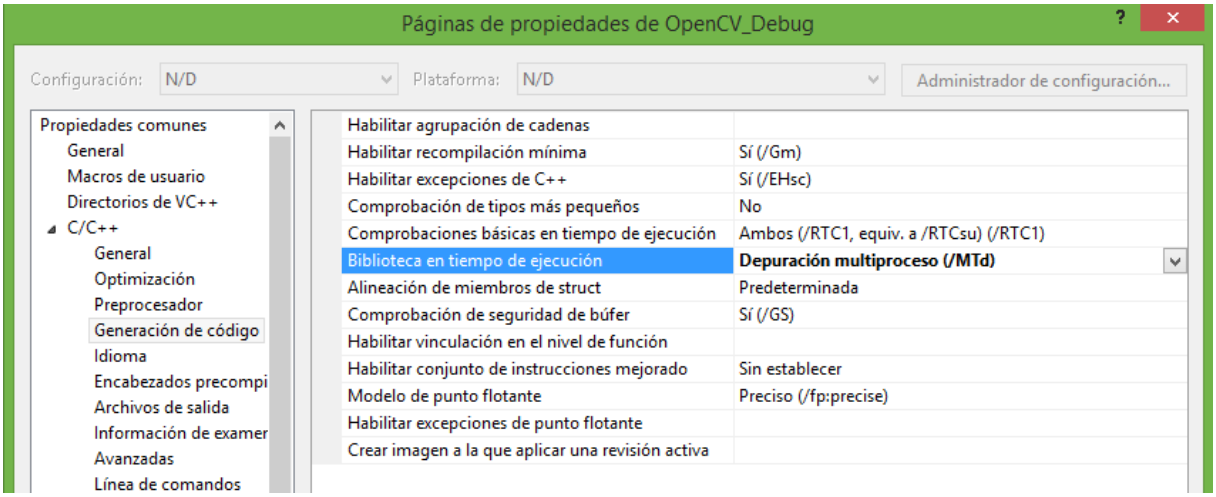


Figura 51 Elegir (/MTd)

Por último, en la hoja de propiedades **OpenCV_Release**, *Propiedades comunes -> C++ -> Preprocesador -> Definiciones de preprocesador* (figura 42), añadir la línea **_ITERATOR_DEBUG_LEVEL=0**.

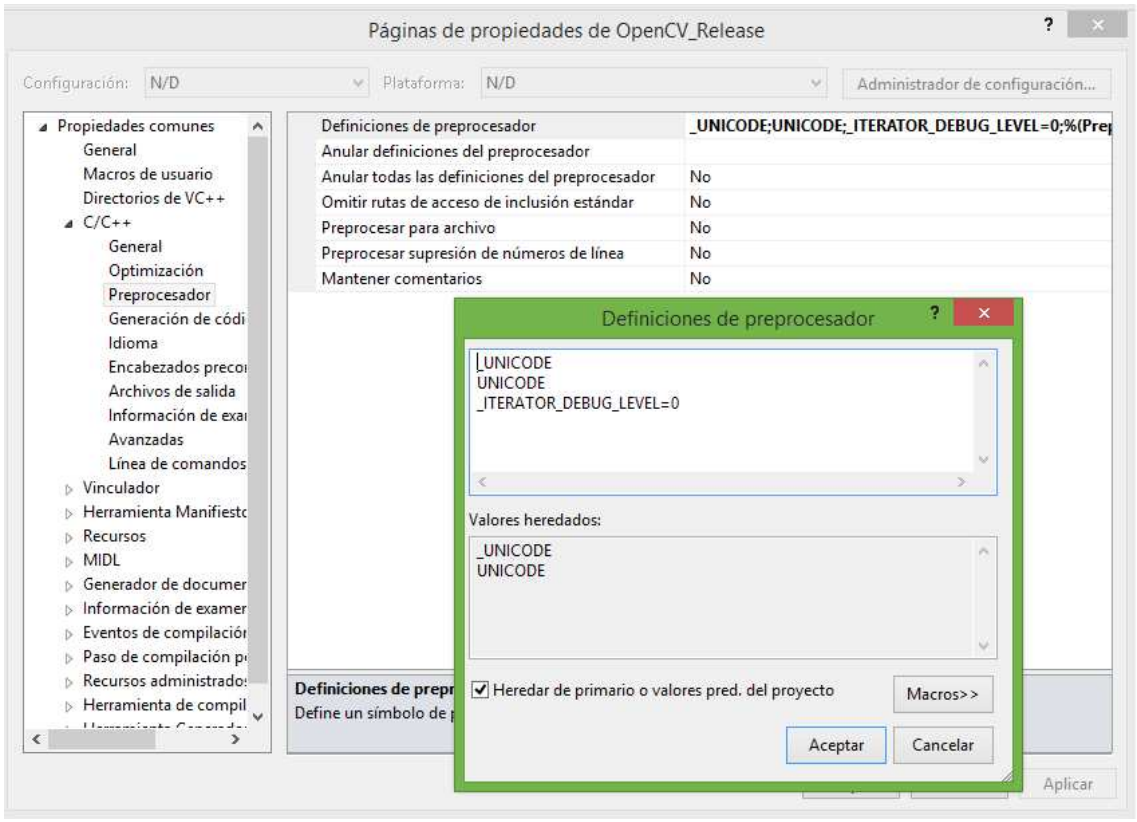


Figura 52 Definiciones de preprocesador

Anexo B. Archivos de cabecera

funcimagen.h

```
#ifndef FUNCIMAGEN_HPP_
#define IMAGENFUNC_HPP_

#include "opencv2\opencv.hpp"

// Busca una cara en la imagen. Si encuentra una o más, rodea con un círculo la mayor
y pone true cara_encontrada.
// Si no encuentra ninguna imagen, devuelve la imagen original y pone a false Cara
Encontrada
cv::Mat CirculoCara (const cv::Mat imagen, const int ancho_min_cara, const int
alto_min_cara, bool& cara_encontrada);

// De una imagen, si encuentra una cara devuelve solo la cara en escala de grises,
ecualizada y escalada al tamaño mínimo
// Si no devuelve un mat vacío
cv::Mat ProcesadoCara(const cv::Mat imagen, const int ancho_min_cara, const int
alto_min_cara);

// De una imagen, devuelve un rectángulo indicando la posición de la cara de mayor
tamaño
// Si no encuentra ninguna cara, devuelve un rectángulo de tamaño 0
cv::Rect EncontrarCara(const cv::Mat frame, const int ancho_min_cara, const int
alto_min_cara);

#endif /* FUNCIMAGEN_HPP_ */
```

funcreconocimiento.h

```
#ifndef FUNCRECONOCIMIENTO_HPP_
#define FUNCRECONOCIMIENTO_HPP_

#include "opencv2\opencv.hpp"
#include <direct.h>
#include <fstream>
#include <conio.h>

class BDCaras {
	public:
		BDCaras();
		// Inicializa el objeto
		void nueva();
		// Carga los atributos leyendo el archivo txt s_archivo_img almacenado en
la subcarpeta s_ruta, devuelve la referencia más alta
		int carga_datos(const cv::String s_ruta, const cv::String s_archivo_img);
		// Abre una ventana y muestra una imagen con las 4 primeras eigenfaces
		void mostrar_eigenfaces ();
		// Añade a la base de datos la persona con nombre s_nombre y almacena en
el disco duro su imagen facial asociada. Devuelve la última referencia
		int agregar_persona(cv::Mat cara, const cv::String s_nombre);
		// Añade a la base de datos la imagen facial pasada y la asocia al
usuario al que pertenece la referencia iRef
		bool agregar_foto(cv::Mat cara, int iRef);
		// En el archivo txt marca las imágenes del usuario iRef como borradas.
Vuelve a aplicar el PCA sn dichas imágenes
		bool borrar_persona(int iRef);
		// Realiza el reconocimiento de la cara pasada. Devuelve la referencia
del usuario o 0 si no encuentra ninguna cara parecida
		int buscar_cara(cv::Mat cara);
		// Igual que la anterior, también añade el nombre de la persona
encontrada
		int buscar_cara(cv::Mat cara, cv::String& nombre);
		// Realiza el test de reconocimiento. Devuelve el número de imágenes en
las que no ha encontrado caras en el entrenamiento
	       int test_reconocimiento (int& iTotal, int& iPositivo, int& iFalsoPositivo,
int &iNegativo);
	      // Devuelve la referencia de la persona pasada. Si no encuentra el nombre,
devuelve 0
		int buscar_persona (const cv::String s_nombre_persona);
		// Devuelve el total de imágenes que tiene el usuario asociado a la
referencia pasada
		int total_imagenes_usuario (int iRef);
		// Sustituye la imagen almacenada en el disco duro por la pasada
		void modificar_imagen (cv::Mat cara, int iRef, int i_imagen);


	private:
		std::vector<cv::Mat> v_caras;
		std::vector<int> refs;
		std::vector<cv::String> v_nombres;
		std::vector<cv::Mat> v_caras_pro;
		int ultima_ref;
		int alto_caras;
		int ancho_caras;
		int total_eigenvectors;
		cv::PCA pca;
		cv::String ruta;
```

```
            cv::String nombre_archivo_img;
            void aplicar_pca ();
            int carga_datos_test(const cv::String s_ruta, const cv::String
s_archivo_img, int & iCaraNoEncontrada);
};

#endif /* FUNCRECONOCIMIENTO_HPP_ */
```